Imperial College London

Master Thesis

Global Optimisation Using Back-Tracking and Stochastic Methods

Author:
Karol Pysniak

Supervisor:
Dr. Panos Parpas

A thesis submitted in fulfilment of the requirements for the degree of Master of Engineering

in the

Department of Computing

June 2013

Declaration of Authorship

I, Karol Pysniak, declare that this thesis titled, 'Global Optimisation Using Back-Tracking and Stochastic Methods' and the work presented in it are my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

IMPERIAL COLLEGE LONDON

Abstract

Department of Computing

Master of Engineering

Global Optimisation Using Back-Tracking and Stochastic Methods

by Karol Pysniak

In this thesis we propose and analyse new back-tracking and stochastic methods applied to global optimization algorithms. The back-tracking methods in global optimization involve using the information gathered by the points that have already been visited in the domain. In this work we introduce an information model based on Gaussian Processes to Multiple Start Search algorithm and add the integral term to Stochastic Gradient Descent algorithm to steer its trajectory away from the explored regions. Stochastic methods in global optimization involve introducing random processes that maintain converging property of algorithms and in this thesis we propose adding Stochastic Term to the candidate point ranking in Hyperbolic Cross Points and introduce new cooling functions for Stochastic Gradient Descent algorithm and prove that those functions do not alter the mathematical properties of the original algorithm.

The performance of the proposed methods is evaluated on three test functions: Pinter function, Michalewicz function and Restrigin function. Those functions are chosen, because they are extremely challenging for global optimization algorithms to find a solution, but can be easily solved by a human. All of the methods are also compared against each other by tracking their performance when applied to Protein Folding Problem based on AB off-lattice model.

Acknowledgements

I would like to thank my supervisor, Dr. Panos Parpas for agreeing to supervise my project and providing me with constant support, motivating and inspiration throughout the course of project. Furthermore, I would like to thank my second supervisor, Dr. Daniel Kuhn for reviewing my draft report. I would also like to thank my personal tutor, Prof. Paul Kelly for support and enthusiasm during my four years at Imperial College London. Finally, I would like to thank my family and friends for their support throughout my time at Imperial College London.

Contents

List of Figures

Dedicated to my parents...

Chapter 1

Introduction

1.1 Project Explanation

The goal of the project is proposing and adding back-tracking and stochastic methods to global optimization algorithms in order to improve the performance of those algorithms by decreasing the number of function evaluations and improving error accuracy, that is finding a solution that is closer to true, global minimum.

Back-tracking in global optimization uses information gathered by points in the domain that have been already visited by trajectory, that is we do not use only the current knowledge contained in the position of trajectory, but also the information gained from visiting historic points and their relations to make a decision on the next step. In this work we introduce information model based on Gaussian Processes to Multiple Start Search algorithm and add the integral term to Stochastic Gradient Descent algorithm to steer its trajectory away from the explored regions.

Stochastic methods in global optimization involve adding random processes to algorithms that maintain their convergence property. Stochastic methods are used to make algorithms less deterministic and increase the probability of exploring less likely trajectories. In this thesis, we introduce Stochastic Term to the candidate ranking in Hyperbolic Cross Points algorithm and proposed using new cooling functions in Stochastic Gradient Descent that do no alter its mathematical properties.

Back-tracking and stochastic methods described in this dissertation are applied to three main classes of global optimization algorithms:

1. **Multi Start Search** - running local-search deterministic algorithms from multiple starting points uniformly sampled from problem domain
2. **Stochastic Gradient Descent** - solving Stochastic Differential Equation which expresses algorithm trajectory [1]

3. **Hyperbolic Cross Points** - evaluating only points in the domain that compose Sparse Grid - approximation of multi-dimensional function [2]

Mathematically, global optimization algorithms are used to find the global minimum of non-linear function:

$$\min_x f(x) \tag{1.1}$$

$$s.t. : x \in \mathcal{F} \tag{1.2}$$

where $\mathcal{F}$ is a feasible set of a problem:

$$\mathcal{F} = \{x \in R^n | l_i \leq x_i \leq u_i\} \tag{1.3}$$

There is a great abundance of problems in such a form in many fields of science, engineering and finance. The analysed methods could be used to optimize the design of computer system, determine the most optimal way of sampling DNA or optimize complex stock portfolios. In many of those applications, the objective function and corresponding constraints are natural to create, for example, in finance we might want to minimize risk and maximize profit given some initial amount of capital. In the biotechnology sector it is essential to analyse novel proteins and the knowledge of how a protein folds gives the opportunity to predict and tune its chemical and biological properties.

Many local search algorithms exist which can find a local optimum for such models, but effective global search algorithms that promise to find a global optimum are just beginning to become available. However, these global optimization algorithms are compromised by an inability to be scaled up to solve practical, large scale optimization problems, for example, financial portfolio needs to analysed very often under rapidly changing and developing market conditions. The focus in this dissertation is put on scalable back-tracking and stochastic methods used in global optimization algorithms that could be improve the accuracy, precision and rate of convergence of the currently existing algorithms.

1.2 Motivation

The form of the problem to be solved in this dissertation is defined as:

$$\min_x f(x) \tag{1.4}$$

$$s.t. : x \in \mathcal{F} \tag{1.5}$$

where:

$$\mathcal{F} = \{x \in R^n | l_i \leq x_i \leq u_i\} \tag{1.6}$$

f is assumed to be twice continuously differentiable.

The generally used method to find the minimum of the function is following the direction of gradient of the function, that is at every iteration we update the value of starting point x_0 in the following way:

$$x_{k+1} = \prod \left[x_k - \alpha_k \nabla f(x_k)\right] \tag{1.7}$$

where $\nabla f(x_k)$ is a gradient direction and α_k is a step size used when following $-\nabla f(x_k)$.

The projection operator $\prod [x_i] = y_i$ is defined

$$\prod [x_i] = \min\left(u_i, \max\left(l_i, x_i\right)\right) \tag{1.8}$$

The described method guarantees only the convergence to the local point of extrema, which does not necessarily need to be global minimum. Therefore, there is a need for global optimization algorithms.

One way to solve that problem with falling into local minima is random sampling of the feasible set $\mathcal{F}$ and following the gradient descent from those points. However, random search must be done very extensively without any guarantee that the global minimum will be found. Another problem with that method is the fact that due to randomness the same region could be explored many times which would result in increasing the time the algorithm takes to find a global minimum. Hence, to tackle that problem, in this dissertation, we are going to propose adding Gaussian Processes to model the explored areas and rate the points based on how much new information could be gained by following randomly sampled starting points. By doing this, we could avoid exploring the already explored regions.

In this dissertation we also consider another global optimization algorithm: Hyperbolic Cross Points. We propose adding Stochastic Term to ranking of candidate points in Hyperbolic Cross Points algorithm. The ranking is used to pick the next point in Sparse Grid, the approximation of objective function to be evaluated. However, the currently used rankings in Hyperbolic Cross Points algorithm make the algorithm deterministic, therefore not available for Monte-Carlo-like or batch simulations, or is even independent of the objective function. Hence, we add Stochastic Term to make it less deterministic and improve its adaptability to objective function.

Another way to overcome the problem of falling into local minimum rather than finding the global minimum is to add stochastic term to Gradient Descent algorithm. Then, the value of x_{k+1} is updated every iteration in the following way:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \sqrt{2T(t)}dB(t)\,, \{x_k, x_{k+1}\} \subset \mathcal{F} \tag{1.9}$$

where $B(t)$ is the standard Brownian motion in $\mathcal{R}^n$. $T(t)$ is defined as annealing schedule and it is usually of the following form:

$$T(t) \triangleq \frac{c}{log(2+t)}\,, c \leq c_0 \tag{1.10}$$

where c_0 is a constant positive scalar.

However, in the presented form, the algorithm uses only the current position of its trajectory x_k to find another point to evaluate x_{k+1}. It means that we discard any information about the points we have already visited. In this dissertation, we propose adding the integral term which could be treated as a bias towards the unexplored regions, that is steering the trajectory away from the explore regions.

Another problem in using Stochastic Gradient Descent is its cooling function:

$$T(t) \triangleq \frac{c}{log(2+t)}, c \leq c_0 \tag{1.11}$$

The function is used as a cooling function, because algorithm is proved to be weakly convergent for that function [3]. However, in this dissertation we propose using other functions, that could change the behaviour of trajectory by altering the variance of 'noise term' $\sqrt{2T(t)}dB(t)$ in stochastic differential equation governing the trajectory. We also prove the convergence of the new, proposed cooling functions.

1.3 Project aims

The aims of this project are as follows:

- Proposing the improvement to the random sampling of starting points for deterministic, local gradient-descent search algorithm using Gaussian Processes and creating the model to annotate the already explored areas and maximize the possible information gain.
- Adding the stochastic term to the conventional, adaptive ranking of candidate points for Hyperbolic Cross Points global optimization method.
- Analysis, potential gains and effects of using unconventional simulated annealing functions for Stochastic Gradient Descent. We prove that the convergence and other mathematical properties of Stochastic Gradient Descent are not compromised by using the proposed cooling functions.
- Introduction of the integral term to Stochastic Gradient Descent Algorithm that would use the information gathered by the past points of trajectories to steer away from the explored regions of the feasibility set
- Analysis of the proposed methods and comparing their performance based on protein folding problem using AB off-lattice model for 2- and 3-Dimensional problem setting with 13 and 21 proteins.

1.4 Report structure

The report is structured in the following set of chapters:

- **Background chapter:** we start the report from giving an introduction to the techniques and methods used in this project. We explain the mathematics necessary to understand that project, for example stochastic calculus and stochastic differential equation, Gaussian processes and their applications. We also present some techniques necessary for numerical solving of ordinary and stochastic differential as well as techniques for numerical integration.

- **Gaussian Processes in Random Starting Points Sampling:** the first proposed improvement to the current method of random sampling of starting points is adding Gaussian Processes to model explored areas of the feasible set (search space). We are going to show the mathematical foundations for adding Gaussian Processes and how they would behave in different situations to optimize the performance and accuracy of deterministic gradient-descent search.

- **Adding Stochastic Term to Hyperbolic Cross Points:** another proposed backtracking improvement involves adding stochastic term to the adaptive ranking used in Hyperbolic Cross Points. We explain the rationale and show that it makes the method non-deterministic, thus making the method more appropriate for, for instance, Monte-Carlo simulations. We also present the performance of the proposed ranking against Adaptive and Non-Adaptive rankings.

- **Simulated Annealing Function Generalization for Stochastic Gradient Descent Algorithm:** in this chapter, we are going to present how different simulated annealing functions (or 'cooling functions') could be found and generalized for Stochastic Gradient Descent Algorithm. We prove that using other annealing functions does not impact on mathematical properties of original Stochastic Gradient Descent Algorithm. We are also going to show how it affects the performance and error accuracy of the original algorithm and compare various functions.

- **Adding Integral Term to Stochastic Gradient Descent Algorithm:** another chapter involves presenting the proposed integral term that uses the information gathered by the past points in order to steer trajectory in Stochastic Gradient Descent Algorithm away from the explored regions. The simulations are run to evaluate how the term affects the trajectory of the algorithm.

- **Global optimization for Protein Folding Problem:** in this chapter, we present AB off-lattice model for protein folding energy. Then, we test all of the proposed methods for global optimization on this problem and compare them against each other based on the achieved results.

- **Conclusion and Future Work:** The final chapters gives both quantitative and qualitative conclusions about the proposed improvements in the existing methods. We also suggest the future works and the natural continuation of the work presented in this dissertation.

1.5 Project Evaluation

We evaluate the results of the project, that is the proposed methods and implementations, at the end of every chapter. To compare methods and performance, we are going to use mathematical functions which are difficult to optimize for a computer, but can be easily solved by a human and are described in the background section. To compare methods, we are going to compare a few statistics:

1. **Number of functions evaluations:** Most of the real-life applications involve functions that are computationally consuming to evaluate. Hence, in most of the applications function evaluations are likely to take the most of the time. Hence, decreasing their number improves the performance of the algorithm.
2. **Error accuracy:** it denotes how close to the true global minimum the achieved result was.
3. **Execution time:** that metric denotes how long a tested algorithm takes.
4. **Number of unique points:** the points are considered to be unique if they are further from each other than some small value ϵ. The higher number of unique points, the more explorative a method is.

Another method we use to evaluate the proposed methods is applying them to the real-life biochemical application and seeing if they could compete with currently existing algorithms and optimization techniques. We evaluate the results of the project not only quantitatively by comparing number of function evaluations or error accuracy, but we also consider the proposed methods qualitatively, that is we explain why some methods might work better than others.

Chapter 2

Background

2.1 Deterministic Optimization Algorithm

The problem to be solved in this dissertation is of the following form:

$$\min_{x} f(x) \tag{2.1}$$

$$s.t. : x \in \mathcal{F} \tag{2.2}$$

where:

$$\mathcal{F} = \{x \in R^n | l_i \leq x_i \leq u_i\} \tag{2.3}$$

$\mathcal{F}$ is a feasible set of a problem. f is assumed to be twice continuously differentiable.

To solve that problem for a convex function, that is a function with only one local minimum, there are many deterministic algorithms. The most basic one is just a simple Gradient Descent Algorithm. The general principle of gradient-descent methods is that we can choose arbitrary, feasible $x_0 \in \mathcal{F}$ and on every iteration update the current x_k in the following way:

$$x_{k+1} = x_k - \alpha_k \nabla f(x) \tag{2.4}$$

However, the biggest problem with that algorithm is that it can easily fall into local minimum, not necessarily global minimum. Another challenging issue is the fact that the value of α_k for each iteration should be such that:

$$\arg\min_{\alpha_k} f(x_k - \alpha_k \nabla f(x)) \tag{2.5}$$

that is, it should minimize the value of $f(x_{k+1})$. However, most of the efficient methods for finding that value might easily miss, or 'slide over', some local minimum.

2.2 Deterministic Gradient Descent - Problem of Falling Into Local Minima

The problem to be solved in this dissertation is of the following form:

$$\min_x f(x) \tag{2.6}$$

The objective function can be of arbitrary non-linear form. Although there are many deterministic ways to solve that problem for linear functions such as, for example, simplex algorithm, there is no method to find a global minimum of arbitrary non-linear function. In such a case, we need to use some other method. We are going to focus on gradient-descent methods. The general principle of gradient-descent methods is that we can choose arbitrary, feasible $x_0 \in \mathcal{F}$ and on every iteration update the current x_k in the following way: The objective function can be of arbitrary non-linear form. Although there are many deterministic ways to solve that problem for linear functions such as, for example, simplex algorithm, there is no method to find a global minimum of arbitrary non-linear function. In such a case, we need to use some other method. We are going to focus on gradient-descent methods.

where α_k is the optimal step-size for a given iteration. The iterations stops when the gradient approaches zero or when the maximum number of iterations has been exceeded. The advantage of this method is the fact that it can be used for any type of function and is easy to implement and calculate. However, the presented approach does not guarantee that it converges to the global minima, but only to a local point of extrema. To show the problem with that technique, let us assume that the function f is of the following form:

$$f = x^2(x-3)(x+2) = x^4 - x^3 + 6x^2 \tag{2.7}$$

To present the problem clearly, two starting points are chosen x_a and x_b:

$$x_a = -0.3 \tag{2.8}$$

$$x_b = 1 \tag{2.9}$$

The gradient of the example function f is:

$$\nabla f(x) = 4x^3 - 3x^3 + 12x \tag{2.10}$$

Hence, we update the value of x every iteration in the following way:

$$x_{k+1} = x_k - \alpha_k(4x_k^3 - 3x_k^3 + 12x_k) \tag{2.11}$$

In the figure below, the starting points and trajectories followed and indicated by gradients have been shown for both cases. The arrows denote the directions indicated by gradients. The starting points for both trajectories and their solutions have also been denoted.

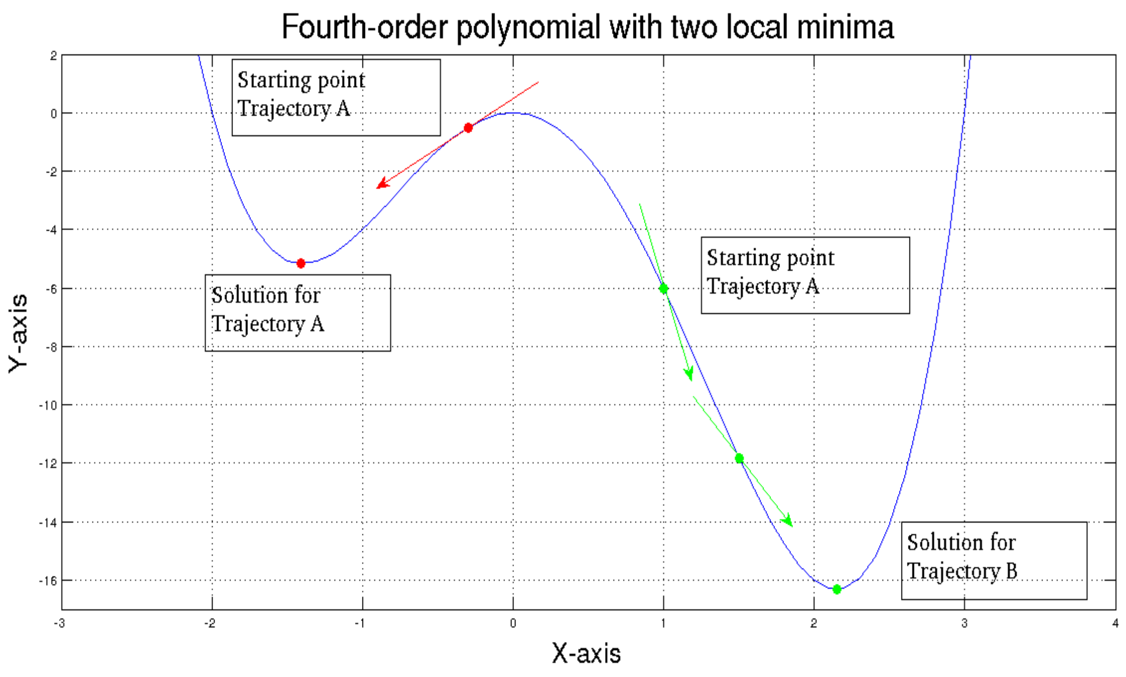

FIGURE 2.1: Gradient descent graph

As can be seen, the solutions obtained by two trajectories are different and are equal to:

$$x_A^* = 1.406 \tag{2.12}$$

$$x_B^* = 2.151 \tag{2.13}$$

The obvious problem with that method that it produces different results depending on what starting point has been picked. Hence, the global minimum could have been missed and, therefore, the method is considered unreliable for global optimization.

To tackle that problem, that is falling into locally optimal minima, it is very common to add 'noise' term that would allow the trajectory not to follow the gradient direction, but try other, random direction. Among many other methods, the problem could be also solved by applying Monte-Carlo simulation and extensively searching a feasible set for a possible solution.

However, currently global optimization is very open problem to research. Due to the increasing number of applications, for example, in finance, biochemistry or machine learning, more and more methods are proposed, but there is no definite way of finding a global minimum which outperforms other methods. Hence, it is very common that new algorithms are proposed with the aim of finding a global minimum in specific application or type of non-linear function.

2.2.1 Newton's Method

The method presented in the previous sections use only the first derivative of our objective function $f(x)$. However, sometimes it might be more efficient use both first and second derivatives. To overcome that drawback, we use Newton's method [4]. The most important fact about Newton's Method is that we can obtain an approximation to twice continuously differentiable function using the Taylor series expansion of f about the current point x_k:

$$f(x) \approx f(x^{(k)}) + (x - x^{(k)})^T g^{(k)} + \frac{1}{2}(x - x^{(k)})^T F(x^{(k)})(x - x^{(k)}) \tag{2.14}$$

Because the terms of order three and higher do not impact the rest of the equations, we have neglected all the terms of third or higher order. Applying the First Order Necessary Condition, which states that for x^* to minimize $f(x)$, the following condition must be satisfied:

$$g(x^*) = \nabla f(x^*) = 0 \tag{2.15}$$

we get:

$$f(x) \approx f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T F(x^{(k)})(x - x^{(k)}) \tag{2.16}$$

According to the Second Order Necessary Condition, for x^* to be a minimizer, it is must be true that:

$$F(x^*) > 0 \tag{2.17}$$

Applying the SONC to the equation (2.6), we get the following iteration, which represents Newton's method:

$$x^{(k+1)} = x^{(k)} - F(x^{(k)})^{-1} g^{(k)} \tag{2.18}$$

To guarantee that Newton's Method has gradient descent property, we can add the step-size α_k to the method:

$$x^{(k+1)} = x^{(k)} - \alpha_k F(x^{(k)})^{-1} g^{(k)} \tag{2.19}$$

where α_k is chosen so that the following equation is satisfied:

$$f(x^{(k+1)}) < f(x^{(k)}) \tag{2.20}$$

The biggest problem with Newton's Method is the fact that it does not always convergence. In fact, for Newton's Method to convergence, some conditions must be satisfied, for example, starting points must be relatively close to the solution and the inverse of second derivative must exists at the solution.

2.2.2 Quasi Newton's Method

Another substantial drawback of Newton's Method is the fact that we need to calculate the inverse of second derivative $F(x)^{-1}$ every iteration, which might be extremely computationally challenging. Quasi Newton Algorithms offer easier ways [5]. Their main iteration is:

$$x^{(k+1)} = x^{(k)} - \alpha_k H_k g(x^{(k)}) \tag{2.21}$$

where H_k is a square real matrix and α_k is the term guaranteeing the optimal descent property. To guarantee the decrease in $f(x)$ at every iteration, the matrix must satisfy the following property:

$$g(x^{(k)T})H_k g(x^{(k)}) > 0 \tag{2.22}$$

or, simpler, H_k must be positive semi-definite.

Now, we are going to present some of the most popular and accurate Quasi-Newton Methods.

2.3 DFP Algorithm

The DFP algorithm, developed by Davidon, Fletcher and Powell in [6] and [7] works as follows:

1. Select starting point x_0 and initial real symmetric positive definite H_0
2. if $\nabla f(x_k) = 0$, stop. Otherwise, $d_k = -H_k \nabla f(x_k)$.
3. Set:

$$\alpha_k = \arg\min_{\alpha} f(x_k + \alpha_k d_k) \tag{2.23}$$

and:

$$x_{k+1} = x_k + \alpha_k d_k \tag{2.24}$$

4. Compute:

$$\Delta x_k = \alpha_k d_k \tag{2.25}$$

$$\Delta g_k = g_{k+1} - g_k \tag{2.26}$$

$$H_{k+1} = H_k + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{(H_k \Delta g_k)(H_k \Delta g_k)^T}{\Delta g_k^T H_k \Delta g_k} \tag{2.27}$$

5. Increment k, go back to step (2) of the algorithm.

2.4 The BFGS Algorithm

Another, widely used Quasi-Newton Method has been developed by Broyden, Fletcher, Goldfarb and Shanno [8]. It is very similar to the DFP Algorithm. It differs by how H_n is calculated. The method used to calculate it is as follows:

$$H_{k+1} = H_k + \left(1 + \frac{\Delta g_k^T H_k \Delta g_k}{\Delta g_k^T \Delta x_k}\right) \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{H_k \Delta_k \Delta x_k^T + (H_k \Delta_k \Delta x_k^T)^T}{\Delta g_k^T \Delta x_k} \tag{2.28}$$

Although the BFGS Algorithm update of H_{n+1} might seem computationally more requiring, it is not as the equation presented has many common terms and, therefore, it might quite efficiently calculated.

It ends the presentation of Deterministic Local Search Algorithms that are going to be used in this dissertation.

2.5 Line Search

Every Deterministic Optimization Algorithm presented so far uses an optimal step-size α_k, that is:

$$\alpha_k = \arg\min_{\alpha} f(x_k + \alpha_k d_k) \tag{2.29}$$

To find the value of α_k, we are going to use the line-search method. The algorithm is as follows:

- Set $k = 0$, $\lambda_l = \lambda_{min}$, $\lambda = \lambda_{max}$
- Set $\lambda_{mid} = \frac{\lambda_l + \lambda_u}{2}$ if $\nabla f(\lambda_{mid}) < 0$, then $\lambda_u = \lambda_{mid}$ if $\nabla f(\lambda_{mid}) > 0$, then $\lambda_l = \lambda_{mid}$
- if $|\nabla f(\lambda_{mid})| < \epsilon$, then stop the algorithm and $\alpha_k = \lambda_{mid}$.

2.6 Stochastic Processes

Stochastic Process is a set of random variables used to represent the behavior of random value or system over time. The system which involves stochastic process exhibits certain undetermined behavior and, in contrary to deterministic systems, it can evolve into many, different states. The format definition is that given a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ and a measurable space $(S, \mathcal{S}$, an S-valued stochastic process is a collection of S-value random variables on Ω, indexed by a totally ordered set T - time.

2.7 Brownian Motion

The Brownian Motion (or the Wiener Process) is a stochastic process W that exhibits the following properties over time $[0, T]$:

1. $W(0) = 0$ with probability 1
2. For $0 \leq s < t \leq T$, the random variable given by the increment $W(t) - W(s)$ is normally distributed, that is $W(t) - W(s) \sim \sqrt{(}t - s)N(0, 1)$ ($\mu = 0$ and $\sigma^2 = t - w$.
3. For $0 \leq s < t < u < v \leq T$, $W(t) - W(s)$ and $W(u) - W(v)$ are independent.

2.8 Stochastic Calculus

Stochastic Calculus is a calculus used to integrate over random variables. There are many examples of such problems in everyday life, for example, let us imagine the following problem: number of server requests grow at a rate $a(t)$. However, there is a noise involved due to the randomness of that process. Therefore, the total rate of growth could be denoted as:

$$\frac{dN}{dt} = a(t) + "noise" \tag{2.30}$$

The standard calculus and Ordinary Differential Calculus does not allow to integrate over the unknown random variables. In general, we can write the problem above as:

$$\frac{X}{t} = b(t, X_t) + \sigma(t, X_t)W_t \tag{2.31}$$

In that equation, W_t denotes the standard Brownian motion.

Let us consider the discrete version of the equation above over $[0, t]$ and let $0 = t_0 < t_1 ... < t_m = t$:

$$X_{k+1} - X_k = b(t_k, X_k)\Delta t_k + \sigma(t_k, X_k)W_k\Delta t_k \tag{2.32}$$

Considering the assumptions stated for the Brownian Motion, we can write that $W_t\Delta t_k = \Delta B_k$. Knowing the initial value for X_0, we can write:

$$X_k = X_0 + \sum_{j=0}^{k-1} b(t_j, X_j)\Delta t_j + \sum_{j=0}^{k-1} \sigma(t_j, X_j)\Delta B_j \tag{2.33}$$

Assuming that some limit exists as $\Delta_j \to 0$, we can apply standard integration notation to get:

$$X_t = X_0 + \int_0^t b(s, X_s)\,\mathrm{d}s + "\int_0^t \sigma(s, X_s)\,\mathrm{d}B_s" \tag{2.34}$$

It has been shown that, in fact, the solution to the right "noisy" integral: $\int_0^t \sigma(s, X_s)\,\mathrm{d}B_s$ exists and can be found using, for example, Ito's Integral or Stratonovich's Integral [9].

2.9 Ito's Integral

It has been proven by Ito that assuming variable S follows an Ito process, that is it contains a stochastic and non-stochastic term or:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{2.35}$$

then the following statement is true:

$$dG(s,t) = \left(\mu S_t \frac{\delta G}{\delta S} + \frac{\delta G}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 G}{\delta S^2} dt\right) + \sigma S_t \frac{\delta G}{\delta S} dW_t \tag{2.36}$$

Then, by transforming the given function to the function of the above form, we can calculate the integral and the solution for a stochastic process. We are going to see the application of that integral on the following, widely-used example.

In finance, a forward contract is priced as follows:

$$F_0 = S_0 e^{rT} \tag{2.37}$$

or in general:

$$F = Se^{r(T_1 - T_2)} \tag{2.38}$$

where: S_0 is a starting price, r is a risk-free rate of growth of capital, T is time and, finally, F_0 is the expected forward price at time T.

Assuming that S is of the following form:

$$dS = \mu S dt + \sigma S dW_t \tag{2.39}$$

we could use Ito's lemma and substitute the Ito's Integral with our current function. Hence, we can write:

$$dG(s,t) = dF(s,T) \tag{2.40}$$

$$dF(s,T) = (\mu S_t \frac{\delta F}{\delta S} + \frac{\delta F}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 F}{\delta S^2})dt + \sigma S_t \frac{\delta F}{\delta S} dW_t \tag{2.41}$$

Now, we need to calculate all the necessary terms:

$$\frac{\delta F}{\delta t} = -rSe^{r(t_1 - t_2)} \tag{2.42}$$

$$\frac{\delta F}{\delta S} = e^{r(t_1-t_2)} \tag{2.43}$$

$$\frac{\delta^2 F}{\delta S^2} = 0 \tag{2.44}$$

Having all the terms above calculated, we substitute them into Ito's Integral and get:

$$dF(s,T) = (\mu S_t e^{r(t_1-t_2)} - rSe^{r(t_1-t_2)} + \frac{1}{2}\sigma^2 S^2 0)dt + \sigma S_t \frac{\delta F}{\delta S} dW_t \tag{2.45}$$

$$dF(s,T) = (\mu Se^{r(t_1-t_2)} - rSe^{r(t_1-t_2)})dt + \sigma Se^{r(t_1-t_2)} dW_t \tag{2.46}$$

$$dF(s,T) = (\mu - r)Se^{r(t_1-t_2)}dt + \sigma Se^{r(t_1-t_2)} dW_t \tag{2.47}$$

Knowing that $F = Se^{r(t_1-t_2)}$, we have:

$$dF = (\mu - r)Fdt + \sigma F dW_t \tag{2.48}$$

However, because the second term on the right-hand side of the equation contains dW_t is a standard Wiener process and its expected value is 0, we know that $\sigma F dW_t \to 0$. Hence, we can write:

$$dF = (\mu - r)Fdt + \sigma F dW_t = (\mu - r)Fdt \tag{2.49}$$

By doing this, we have shown that the expected growth of the forward price, that is the difference between the current and future price of a stack, is proportional to $(\mu - r)F$.

2.10 Stratonovich Integral

There is also another way of solving the problems described in the previous sections. It has been created by Stratanovich, independent of Ito and his calculus. Although that integral is not used in this dissertation, it is important to mention that there are many different approached to the considered problem. The main different between Ito's Integral and Stratonovich Integral is the fact that using Stratonovich Integral allows us to apply chain's rule when manipulating equations. However, they are equivalent in most of the cases and can be converted to each other when convenient.

2.11 Stochastic Differential Equations

Stochastic Differential Equations (or SDE in short) is a differential equation which has at least one stochastic term. The general form of such an equation is:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{2.50}$$

The example of such an equation was the forward pricing problem given in the previous paragraph. However, the differential equation we are going to focus on and try to solve in this dissertation is as follows:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \sqrt{2T(t)}dB(t)\,, \{x_k, x_{k+1}\} \subset \mathcal{F} \tag{2.51}$$

where $B(t)$ is the standard Brownian motion in $\mathcal{R}^n$. $T(t)$ is defined as annealing schedule and it is usually of the following form:

$$T(t) \triangleq \frac{c}{log(2+t)}\,, c \leq c_0 \tag{2.52}$$

However, that equation does not have a close-form solution. Therefore, we are going to use various numerical experiments to solve that problem.

2.12 The Euler-Maruyama Method

One of the most popular methods to approximate the solution for a stochastic differential equations is The Euler-Maruyama Method. It generalizes the Euler method used for solving an ordinary differential equation. It assumes the following form of a Stochastic Differential Equation:

$$dX_t = a(X_t)dt + b(X_t)dW_t \tag{2.53}$$

where X_t is the solution. dW_t is a standard Wiener Process.

Knowing the initial X_0 and the time interval $[0, T]$ which the Stochastic Differential Equation is to be integrated over, we start from dividing time interval $[0, T]$ into N intervals so that:

$$\Delta t = \frac{T}{N} \tag{2.54}$$

Then, we recursively calculate the value of X_i until all intervals have been calculated:

$$X_{i+1} = X_i + a(X_i)\Delta t + b(X_i)\Delta W_i \tag{2.55}$$

The values of ΔW_i are simulated by random variables, independent of each other and normally distributed with the mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\Delta t}$, where $\Delta t = \frac{T}{N}$.

To show the usefulness and simplicity of that method, we are going to analyze the following example of pricing a forward price of a stock. The formula that models the price of a stack is of the following form:

$$dF = \mu dt + \sigma dW \tag{2.56}$$

that is, the price of a stock is influenced by two factors: constant change (μ) and some random noise. According to Ito's Integral, the solution to that problem is:

$$F(t) = F_0 e^{\sigma W(t) + (\mu - \frac{\sigma^2}{2})t} \tag{2.57}$$

where F_0 is an initial value of a stock. W_t is a noise, t is time and σ is a standard deviation of the noise. The initial value $F(0)$ is 100. However, we could solve that problem using the Euler-Maruyama method. The main iteration in that method is as follows:

$$F(t+1) = F(t) + \mu \Delta t + \sigma \Delta W_t \tag{2.58}$$

$$F_0 = 100 \tag{2.59}$$

The figure 2.2 below shows the result of running three simulations. As can be seen, each simulation produced different values and traces. To capture the general trend of some function, we could run many Monte-Carlo-like simulation and average the results.

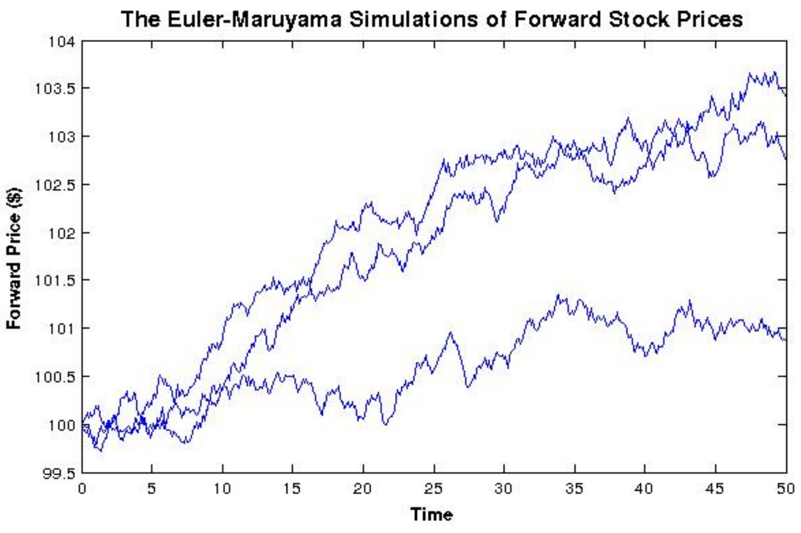

FIGURE 2.2: The Euler-Maruyama Simulation of Forward Stock Prices

2.13 Numerical Integration

To evaluation any integral that is hard to solve analytically or is not continuous, we can use numerical integration methods. In general, we have the following integral to solve:

$$F(a,b) = \int_b^a f(x)dx \tag{2.60}$$

In this dissertation, we are going to use the following method for integral evaluation:

$$F(a,b)\int_a^b f(x)dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2}+\sum_{k=1}^{n-1}\left(f\left(a+k\frac{b-a}{n}\right)\right)+\frac{f(b)}{2}\right) \tag{2.61}$$

2.14 Gaussian Processes

Gaussian Process is a stochastic process used in probability and statistics. It is an important and useful tool for simulation and modeling. In general, Gaussian Process gives a value for a set of random variables such that each of the variables is normally distributed with its own mean μ_i and standard deviation σ_i. To model a Gaussian Process, we need to choose a Gaussian basis function. The functions are usually problem-dependent and are found mostly empirically. The basis function used in this dissertation is:

$$\phi_j = exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.62}$$

In general, the Gaussian Process for n-dimensional problems could be defined as:

$$\phi_j = exp\left(-\sum_{j=1}^{n}\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right) \tag{2.63}$$

The very important application of Gaussian Processes is creating models. The models are described as the sum of Gaussian Processes, that is:

$$W(x) = \sum_{j=1}^{n}\phi_j \tag{2.64}$$

and using our Gaussian basis function, we have:

$$W(x) = \sum_{j=1}^{n} exp\left(-\sum_{j=1}^{n}\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right) \tag{2.65}$$

The figure below shows the example set of Gaussian Processes. That method could be used to, for example, model the prices of flats or financial assets when there are many factors that impact the final price of the product:

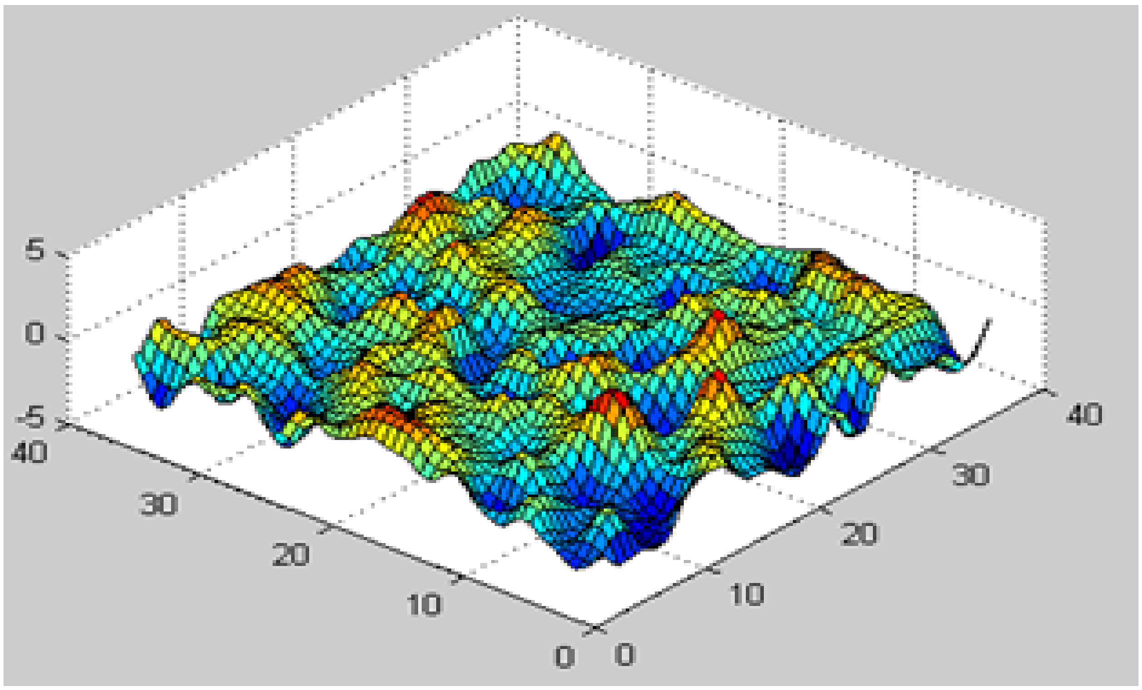

FIGURE 2.3: Sum of Two-dimensional Gaussian Processes

2.15 Simulated Annealing

To overcome the problem of falling into local minima when using deterministic gradient descent in $f(x)$, that is to find such x in the feasible set $\mathcal{F}$ so that:

$$\forall \hat{x} \in \mathcal{F}\, f(x) \leq f(\hat{x}) \tag{2.66}$$

we can add some randomness when deciding how x_k should be changed into x_{k+1}. Instead of strictly following the gradient direction (for unconstrained case):

$$x_{k+1} = x_k - \alpha_k \nabla f(x) \tag{2.67}$$

we can add some noise when updating the value of x_k:

$$x_{k+1} = x_k - \alpha_k \nabla f(x) + \text{"}noise\text{"} \tag{2.68}$$

By doing this, we can escape from local minima. Another advantage is that we can explore regions that would never have changes to be explored. However, the problem with that approach is that we can miss minima that would be explored when using deterministic approach. Because of this, the magnitude of noise should gradually decrease or be updated so that at some point deterministic optimization and gradient direction could be followed. The methods to control that process of 'cooling' annealing is called *simulated annealing*. It is a reference to physical processes of metal annealing - with time the metal is colder and colder and the random movements are less likely. In this dissertation, simulated annealing is used and the direction followed on every iteration is:

$$x_{k+1} = x_k - \alpha_k \nabla f(x) + \sqrt{2T(t))}dB(t) \tag{2.69}$$

Hence, the magnitude and direction of *noise* is:

$$\sqrt{2T(t)}dB(t) \tag{2.70}$$

where t is time and

$$T(t) = \frac{c}{log(2+t)} \tag{2.71}$$

As can be seen above, the function $T(t)$ decreases as t increases and, hence, the magnitude of 'noise' is gradually decreased and, then, deterministic optimization is followed at the best point found so far.

2.16 Optimization Under Constraints

It is very often that the solution to our minimization problem should satisfy certain constraints, for example, we might have budget constraints when optimizing portfolio of stocks or other financial products. Another possible constraints might be time when, for instance, minimizing the most profitable way of delivering services. In general, we might write our problem as:

$$\min f(x) \tag{2.72}$$

$$s.t\, Ax = b \tag{2.73}$$

In such a case, following the gradient descent:

$$x_{k+1} = x_k - \alpha_k \nabla f(x) \tag{2.74}$$

might obviously lead us beyond the feasible set, that is such a set $\mathcal{F}$:

$$\forall x \in \mathcal{F}\, Ax = b \tag{2.75}$$

To make sure that we do not leave the feasibly set, we need to extra term to the direction we follow:

$$x_{k+1} = x_k - P\alpha_k \nabla f(x) \tag{2.76}$$

where:

$$P = I - A^T(AA^T)^{-1}A \tag{2.77}$$

The term P guarantees us that we stay in the feasible set $\mathcal{F}$, assuming that x_0 is in the feasible set, because:

$$A(x_k - \alpha_k P \nabla f(x)) = Ax_k - \alpha_k AP\nabla f(x) = b \tag{2.78}$$

Therefore, using the described technique, we can make sure that our solution complies with the constraints.

2.17 Many-dimensional Test Functions

In this dissertation, we are going to analyze the performance of global optimization techniques on a few functions and some real-life applications. The motivation to choose the mathematical functions presented below is that all of them have a number of local minima and it is very difficult to find their solutions numerically. However, we can set their global minima by changing given parameters. In general, it is for a human to find their global minima, but difficult for a computer.

2.17.1 Pinter's Function

The first function $f(x)$ for multi-dimensional vector x, where x^* is the desired global minimiser, is:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1 P_1(x)) + \sin^2(g_2 P_2(x)) \tag{2.79}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{2.80}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{2.81}$$

For that function $f(x)$, we need to calculate the gradient of the function so that we could use it in gradient-descent optimization techniques:

$$\frac{\delta f(x)}{\delta x_i} = 2s(x_i - x_i^*) + 2g_1\frac{\delta P_1(x)}{\delta x_i}\sin(g_1P_1(x))\cos(g_1P_1(x)) + 2g_2\frac{\delta P_2(x)}{\delta x_i}\sin(g_2P_2(x))\cos(g_2P_2(x)) \tag{2.82}$$

$$\frac{\delta P_1(x)}{\delta x_i} = 2(x_i - x_i^*) + 1 \tag{2.83}$$

$$\frac{\delta P_2(x)}{\delta x_i} = 1 \tag{2.84}$$

The figure below shows what the function looks for one and two-dimensional cases. As can be seen, there are numerous local minima, but global minimum can be easily spotted and specified by a human:

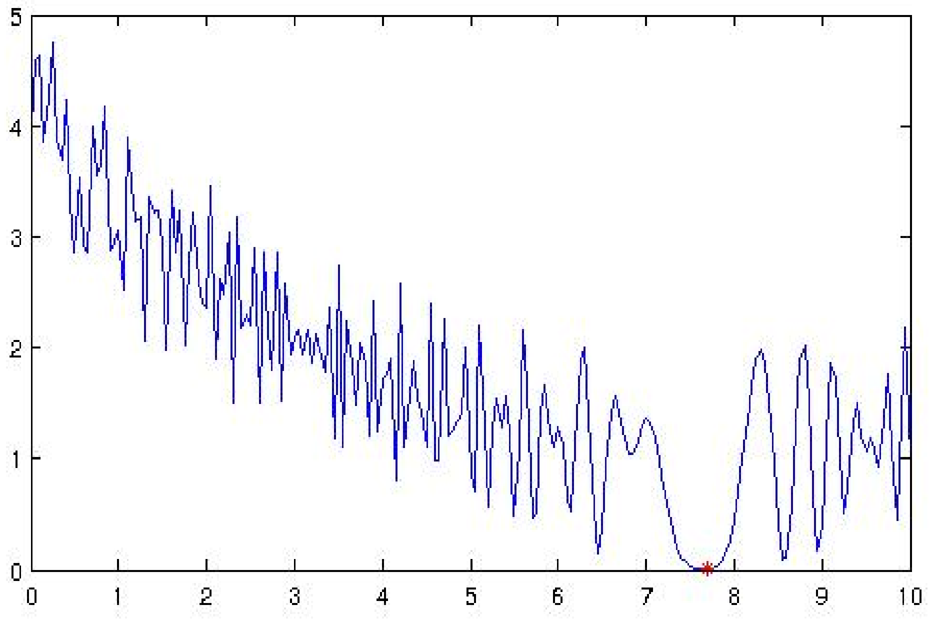

Figure 2.4: Pinter Test Function

2.17.2 Michalewicz function

Another test function we are going to use in this dissertation is Michalewicz Function. For n-dimensional case, it is of the following form:

$$f(x) = \sum_{j=1}^{n} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{2.85}$$

Now, to use that function in our gradient-descent optimization techniques, we need to calculate its gradient, which is:

$$\frac{\delta f(x)}{\delta x_i} = \cos(x_i \sin^{2m}(\frac{ix_i^2}{\pi}) + 4m\frac{i^2 x_i}{\pi^2} \sin(x_i) \sin^{2m-1}(\frac{ix_i^2}{\pi}) \cos(\frac{ix_i^2}{\pi}) \tag{2.86}$$

The Michalewicz function is a multi-modal function with $n!$ local minima.The larger m parameter is, the more steep valleys are created and the more difficult the search for a global minimum is.

The figure below presents Michalewicz function for two-dimensional case:

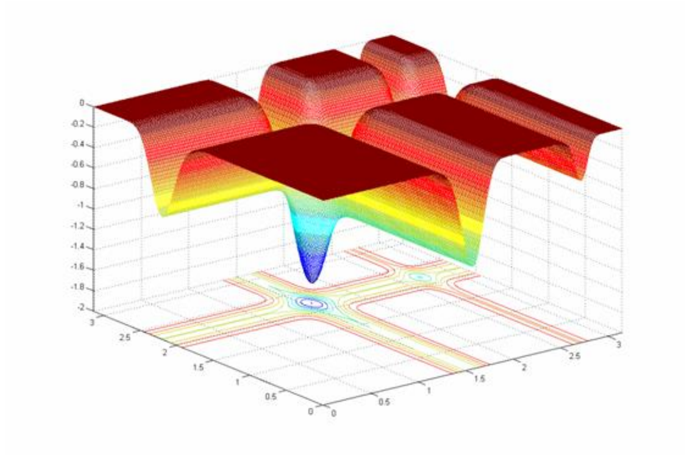

FIGURE 2.5: Michalewicz Test Function

2.17.3 Rastrigin Function

Rastrigin function is the non-convex function of the following form:

$$f(x) = An + \sum_{j=1}^{n}(x_i^2 - A\cos(2\pi x_i)) \tag{2.87}$$

Its gradient is:

$$\frac{\delta f(x)}{\delta x_i} = 2x_i + 2A\pi \sin(2\pi x_i)) \tag{2.88}$$

The graph of that function is:

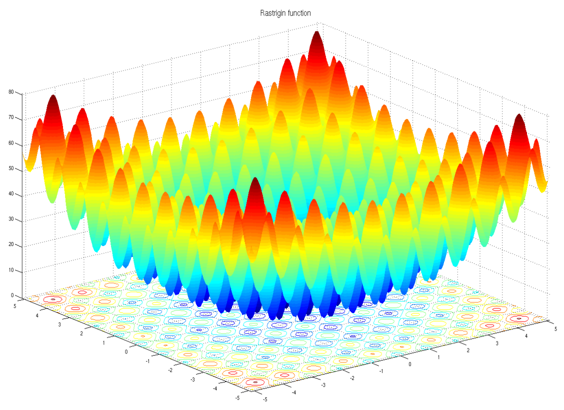

FIGURE 2.6: Rastrigin Test Function

When $A = 10$, the global minimum of that function is $x = 0$ and the value of that function is $f(x) = 10$.

Chapter 3

Gaussian Processes in Random Starting Points Sampling

3.1 Deterministic Gradient-Descent Algorithm

Deterministic Gradient-Descent Algorithm is used to optimize non-linear functions, that is it finds such x that:

$$\min_x f(x) \tag{3.1}$$

The objective function is defined as $f : \mathcal{R}^n \to \mathcal{R}$ and f is assumed to be twice continuously differentiable. In this dissertation, the feasible set is defined using box constraints:

$$\mathcal{F} = \{x \in R^n | l_i \leq x_i \leq u_i\} \tag{3.2}$$

The algorithm starts from choosing the starting point that is in the feasible set $\mathcal{F}$, that is:

$$x_0 \in \mathcal{F} \tag{3.3}$$

In general, every iteration the value of x is updated by the following equation:

$$x_{k+1} = \prod [x_k - \alpha_k \nabla f(x_k)] \tag{3.4}$$

where $\nabla f(x_k)$ is a gradient direction and α_k is a step size used when following $-\nabla f(x_k)$.

The projection operator $\prod [x_i] = y_i$ is defined as:

$$\prod [x_i] = \min(u_i, \max(l_i, x_i)) \tag{3.5}$$

The algorithm stops when $|x_{k+1} - x_k| < \epsilon$, that is we are very close to the point of extrema.

3.2 Random Sampling of Search Space

Although Deterministic Optimization Algorithm finds a minimum, there is no guarantee that the minimum is in fact the global minimum, and not only local one. To overcome that problem, Random Sampling of Search Space is used [10]. It works by randomly choosing any starting point x_0 that belongs to the feasible set $\mathcal{F}$. Then, the gradient-descent algorithm is used and the minimum it reaches is taken note of. The algorithm usually stops when the satisfying point of minimum is found or when the number of iterations exceeds the maximum number. However, the algorithm does not necessarily solve the problem of falling into a global minimum as it might miss the convexity area that could lead to such a point.

3.3 Re-exploring the explored regions

One of the weaknesses of the algorithm described in the previous paragraph is the fact that it might repeatedly choose the points that leads to the same local minimum. It stems from the fact that certain regions, for example regions, which are not very steep, but are convex over big area, are more likely to be explored than small, but very steep regions. To explain that problem more clearly, we are going to analyze the example of Pinter function defined over the domain: $[0, 30]$.

The Pinter function f, defined over $[0, 30]$, is of the following form:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1P_1(x)) + \sin^2(g_2P_2(x)) \tag{3.6}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{3.7}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{3.8}$$

The values of the variables are:

$$s = 0.05; \tag{3.9}$$

$$g_1 = 20; \tag{3.10}$$

$$g_2 = 10; \tag{3.11}$$

The graph of the function is:

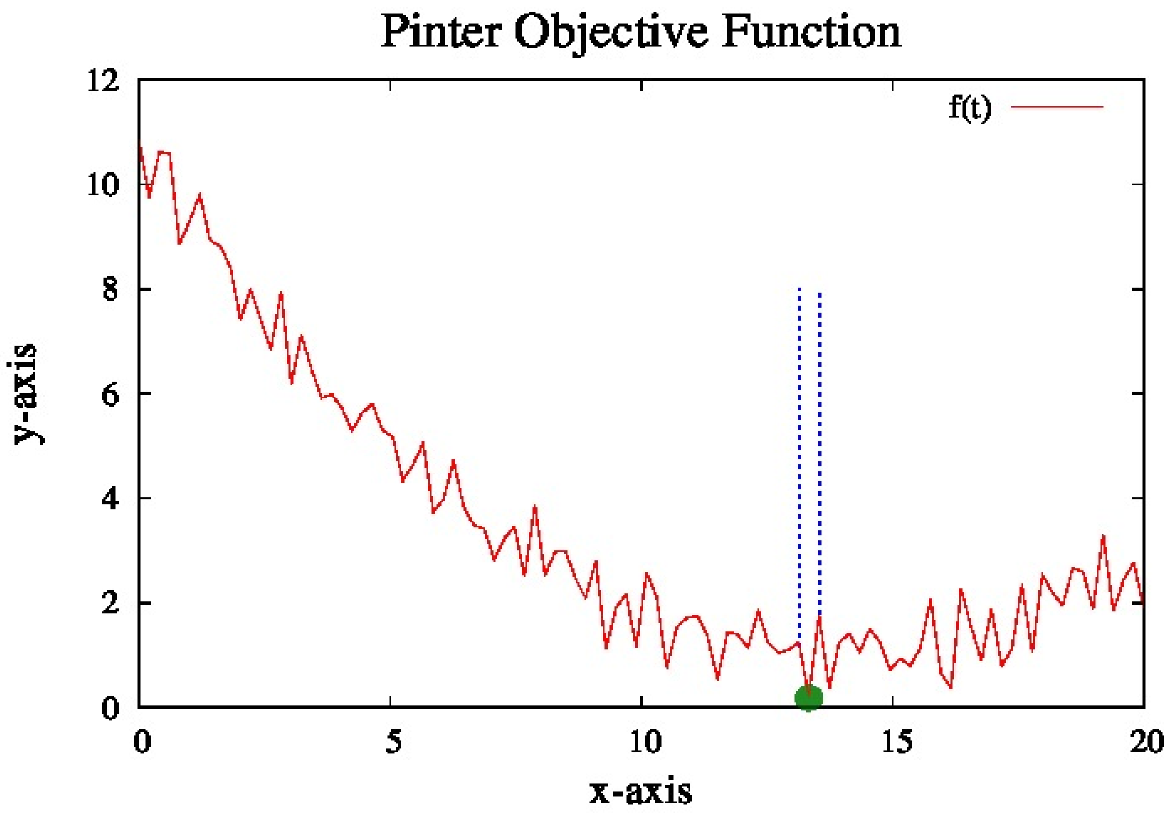

FIGURE 3.1: Function graph

In the figure above, the red line marks the Pinter function with the optimal solution at $x = 14$ with value $f(x) = 0$. The green circle denotes the optimal solution. The two dashed lines mark the region of the domain that, if a starting point is sampled from that region, the global minimum will be found. Mathematically, the algorithm would find the global minimum if the starting point is: $13.15 < x_0 < 13.55$. However, the starting point is uniformly sampled from the interval $[0, 20]$ and, hence we have the probability of $\frac{13.55-13.15}{20-0} = \frac{0.40}{20} = 0.02$ of finding the true, global minimum while the probability of 0.98 of finding some other local minimum. After some number of iterations, the algorithm would find the global minimum, but it could take long time as it would be constantly 'attracted' to any other convexity region with only local minimum. Although in this simple toy example, the problem is not severe, in bigger problems with many local minima the problem becomes significant.

In general, it would be much faster and more accurate for the algorithm to avoid exploring the already explored areas and prefer less likely, but not yet explored areas.

3.4 Gaussian Processes Used in Algorithm

Before explaining how Gaussian Processes are going to be used to solve the described problems, we are going to present what Gaussian Processes used in our solution are going to look like.

The proposed idea to solve the problems described in the previous paragraphs involves adding Gaussian Processes to Random Sampling of Search Space for Deterministic Gradient-Descent Algorithm. As has been shown in the background section, a Gaussian Process could be viewed as a multi-dimensional realization of normal distribution with different means μ_i and standard deviation σ_i for each dimension i. In our application, we are going to define a single Gaussian Process, that is the value of $f(x)$ for some Gaussian Process at point x, as:

$$GP(x) = \exp\left(\sum_{j=1}^{n} \phi(x_j)\right) \tag{3.12}$$

where x_j is jth component of vector x and $\phi(x_j)$ is a kernel function.

Depending on how Gaussian basis function is defined, the properties of Gaussian Process modeled differ. In our solution, we use following Gaussian basis function:

$$\phi(x_j) = \frac{(\mu_j - x_j)^2}{2\sigma_j^2} \tag{3.13}$$

where μ_j is the mean of jth component and σ_j is its standard deviation. The given basis function results in the following value of our Gaussian Process for point x:

$$GP(x) = \exp\left(\sum_{j=1}^{n} \phi(x_j)\right) = \exp\left(\sum_{j=1}^{n} \frac{\mu_j - x_j}{2\sigma_j^2}\right) \tag{3.14}$$

3.5 Information Model Based on Gaussian Processes

As has been described in the previous paragraphs, preventing Random Search Space Sampling for Deterministic Gradient-Descent Algorithm from exploring the already explored regions or at least decreasing the probability of such trajectories, could improve the accuracy of the algorithm and its rate of convergence.

Our proposed solution involves adding tracking and determining which area has been roughly explored by some trajectory. Then, based on this information, we create a Gaussian Process that behaves as centers of 'anti-attraction' and decreases the probability of any trajectory to go in that region. That process could be viewed as creating a model that indicates which areas of the feasibility set are not likely to improve the current solution. To present that idea more clearly, we are going to show some examples how Gaussian Processes are going to be used. The creation and placement of Gaussian Processes will be explained in the next sections.

The first example is going to be based on the function and trajectories described in the section 3.4, when Trajectory Ranking Problem has been presented. The function is:

$$f(x) = x(5x - 1)(x + 2)(x - 1.5) \tag{3.15}$$

The domain of the function is $[-2, \infty]$.

The graph of the function, and potential starting points are presented in the graph below:

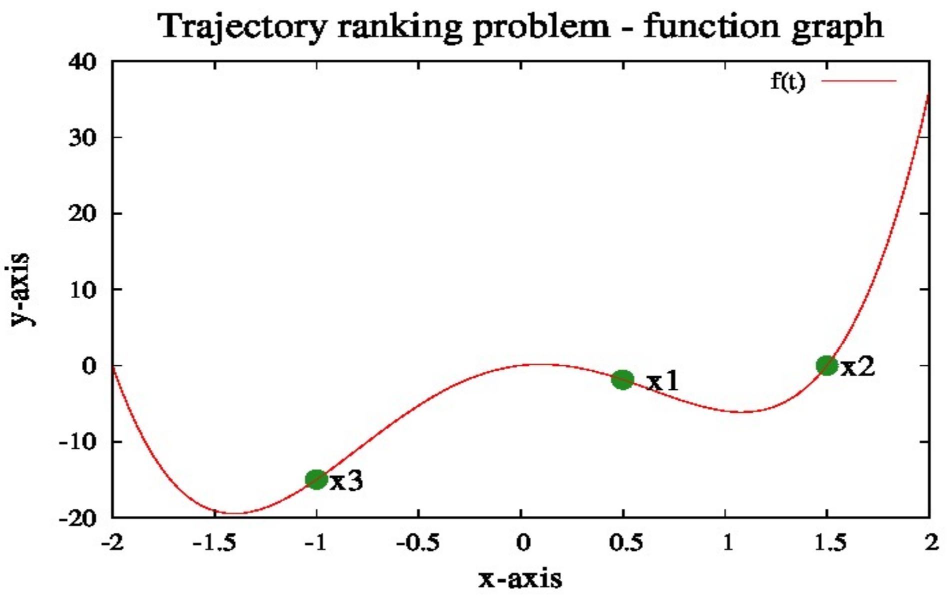

FIGURE 3.2: Trajectory Ranking - Function Graph and Starting Points

The global minimum is at $x =\approx -1.6$. However, any point chosen in the region at $[0, \infty]$ and its local minimum is much more likely to be explored, decreasing the probability of finding the true global minimum. The first points evaluated are x_1 and x_2. The following graph represents the final positions of the trajectories achieved by the algorithm. The green circles labeled x_1 and x_2 represent the final position reached by the trajectories of the corresponding starting points.

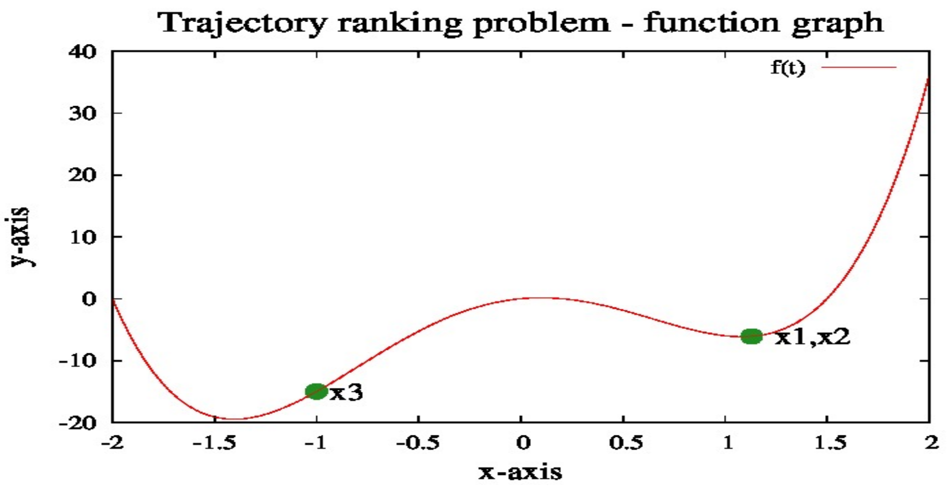

FIGURE 3.3: Final Trajectories

The following graph presents the possible Gaussian Process that could be added as a result of running the algorithm for the described starting points:

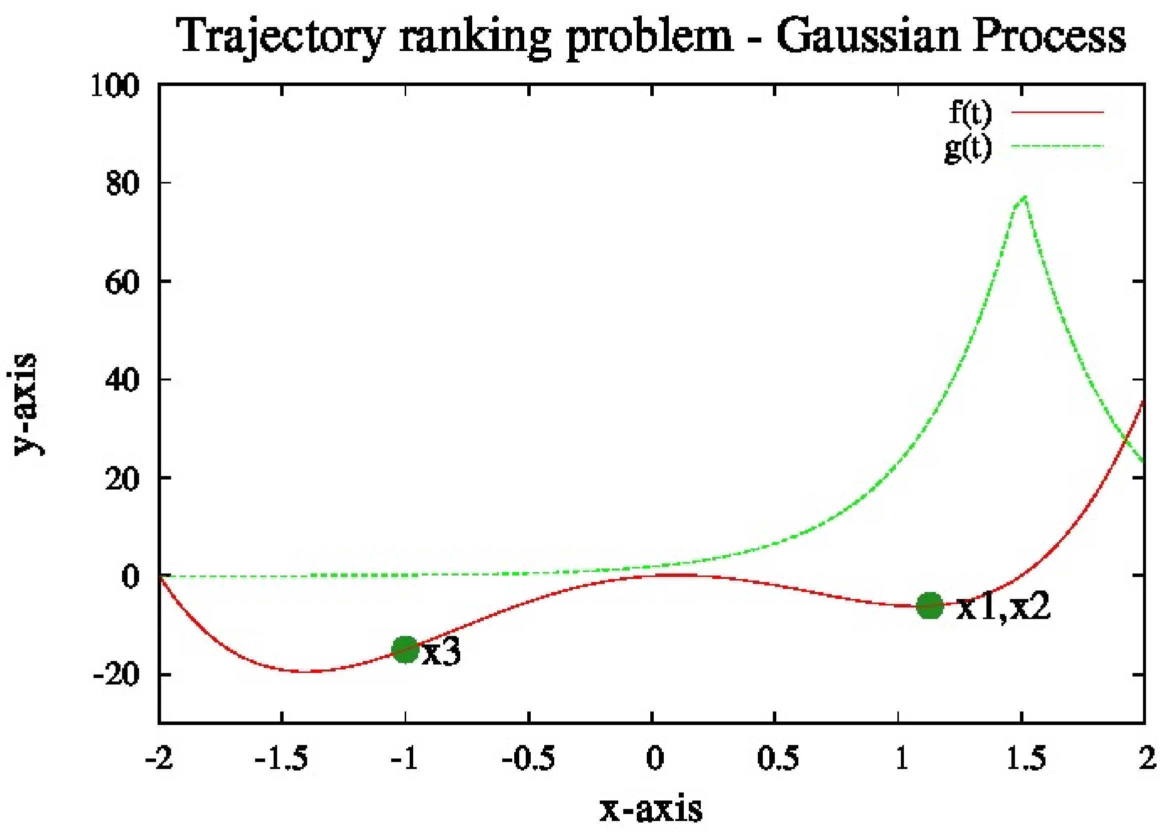

FIGURE 3.4: Possible Gaussian Process added as a result of running the algorithm

The green line denotes the added Gaussian Process after the algorithm terminates for the first two starting points. The higher the value returned by the model (the lower plot in the figure above), the less likely some trajectory could improve the current state or any new information.

As can be noted in the figure above, the convexity area where most of the trajectories are heading has already been explored and Gaussian Process has been added there. The area where the global minimum can be found is not covered by any Gaussian Process and, therefore, the trajectory x_1 which could potentially lead to the global minimum is likely to be chosen over any other trajectories.

3.6 Gaussian Processes in Practice

Using Gaussian Processes that are created and modeled out of data from tracking trajectories, we could change the process of determining the value of most optimal starting points. After randomly drawing some number of potential values for x_0 from a search space, we could use Gaussian Processes in our model to asses them and see if any of the given starting points is likely to add any new information. We obtain that value by calculating the sum of Gaussian Processes at some point and rank them by that value.

The algorithm with the proposed improvement involves:

- **Step 1: Random Sampling of k points** We would start the algorithm from randomly choosing k points from the search space $\mathcal{F}$.
- **Step 2: Starting Points Ranking** We would calculate the sum of Gaussian Processes for all the sampled starting points and sort them by that value in ascending order.
- **Step 3: Choosing most optimal points** We would drop all the points except for the first n points.
- **Step 4: Running the algorithm** We would run Deterministic Gradient Descent Algorithm for all the starting points, that is we would follow the gradient direction:

$$x_{k+1} = x_k - \alpha_k \nabla f(x) \tag{3.16}$$

- **Step 5: Finalizing the iteration** We would take the note of the final points reached by the trajectories and chose the most optimal one.
- **Step 6: Adding Gaussian Processes** We would take note of the trajectory paths and add Gaussian Processes if some trajectory lead through the unexplored region.
- **Step 7: Repeating Iteration** We would repeat the algorithm after the maximum amount of iterations has been reached.

Trajectory x will be evaluated using the following equation:

$$f(x) = \sum_{i=1}^{n} w_i \, GP_i(x) \tag{3.17}$$

where GP_i is ith Gaussian Process for our Search Space and is expressed as:

$$GP_i(x) = \sum_{j=1}^{n} \phi_j \left(x_j, \mu_j, \sigma_j\right) \tag{3.18}$$

where ϕ_j is Gaussian Basis Function, μ_j is a mean and σ_j is a standard deviation for that particular Gaussian Process. w_i is the weight of Gaussian Process GP_i. It is used in order to differentiate the impact of difference Gaussian Processes, especially those which have been merged.

3.7 Adding Gaussian Processes

After choosing the starting points, their gradients are followed by the algorithm. Every time it happens the note is taken of the points visited during that process. Then, the time series of that is created. From that data, we calculate the mean μ and standard deviation σ.

Having the value of mean μ, we need to decide now if a new Gaussian Process should be created or if it should be merged with another one. We are going two use the following condition to determine that:

$$|\mu_i - \mu| < \epsilon \tag{3.19}$$

where μ_i is a mean of any other Gaussian Processes. It means that if there is any other Gaussian Process which has mean μ_i very close to our mean μ calculated from the latest data points, then we should merge those Gaussian Processes as they are most likely to belong to the same convexity area.

If the condition indicates that Gaussian Process should be in fact added, then using the mean μ and σ, we have all the data necessary to create it and place in the correct position.

3.8 Merging Gaussian Processes

If the condition for merging Gaussian Process is satisfied, then we need to use those data to calculate new mean μ and standard deviation σ. There are a rich variety of the ways which can be used to solved that problem and are presented in [11].

However, in order to efficiently merge two Gaussian Processes, we need to hold certain meta-data about each Gaussian Process. The most optimal way to hold those data would be storing all the time series or at least some part of them so that new time series could be just 'incorporated' into the existing and new mean μ and standard deviation σ could be calculated.

That approach posses one, significant problem: its complexity might grow very quickly and storing so many data points would take a lot of memory, therefore such an approach could efficiently be used for small problems. We use that method and try to find the boundary on the size of a problem for which that approach is efficient and tractable.

Another approach for bigger problems is holding only the mean of Gaussian Process μ and its standard deviation σ,. In that approach, we could average the means of two Gaussian Processes so that the new mean μ would simply be:

$$\mu = \mu_i + \mu_j \tag{3.20}$$

where μ_i and μ_j are the means of merged Gaussian Processes.

The proposed approach to obtain the value of σ using the standard deviations of two, merged Gaussian Processes σ_i and σ_j is

$$\sigma = \sqrt{\sigma_i^2 + \sigma_j^2} \tag{3.21}$$

3.9 Merging weights of Gaussian Processes

Our model consists of the following function for trajectory x:

$$f(x) = \sum_{i=1}^{n} w_i \, GP_i(x) \tag{3.22}$$

When we merge two Gaussian Processes with weights w_i and w_j, the impact of the merged Gaussian Process should be bigger. Therefore, we propose the method to combine weights w_i and w_j into new weight w:

$$w = w_i + w_j \tag{3.23}$$

3.10 Deterministic Optimization Algorithm

In our implementation, we use Quasi-Newton method as a Deterministic Optimization Algorithm. In particular, we use the BFGS Algorithm. The difference between the BFGS Algorithm and standard Newton method is that the main iteration is:

$$x_{k+1} = x_k - \alpha_k F_k(x_k)^{-1} \nabla f(x_k) \tag{3.24}$$

where $F_k(x_k)$ is a second derivative of our objective function $f(x)$.

The BFGS Algorithm approximates the value of $F_k(x_k)^{-1}$ as $H(x_k)$ so that the main iteration is now:

$$x_{k+1} = x_k - \alpha_k H_k(x_k) \nabla f(x_k) \tag{3.25}$$

where H_k is defined as:

$$H_{k+1} = H_k + \left(1 + \frac{\Delta g_k^T H_k \Delta g_k}{\Delta g_k^T \Delta x_k}\right) \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{H_k \Delta_k \Delta x_k^T + (H_k \Delta_k \Delta x_k^T)^T}{\Delta g_k^T \Delta x_k} \tag{3.26}$$

Another matter is finding the optimal value of step size α_k. We use line search method to find the optimal condition is:

$$\alpha_k = \arg\min_{\alpha} f(x_k - \alpha_k H_k(x_k) \nabla f(x_k \tag{3.27}$$

3.11 Termination Conditions

As has been mentioned earlier, adding Gaussian Processes to Random Sampling of Search Space for Deterministic Gradient Descent Algorithm has a few termination conditions. They are as follows:

- The algorithm terminates if the total number of iterations exceeds the maximum amount N.
- It also terminates when after M consecutive resets, there has been no improvement in the minimum value of $f(x)$ encountered.

3.12 Evaluation Methodology

To evaluate the algorithm and the proposed change we measure two main statistics for each simulation:

- Number of iterations after which algorithm terminates
- Execution time

We take note of those two statistics after the algorithm achieves the accuracy error of 10 and 1 within the true global optimum. The accuracy error is defined as:

$$error = |x^{'} - x^{*}| \tag{3.28}$$

where x^{*} is a true global minimum and $x^{'}$ is an experimentally found solution. The smaller the error is, the more accurate algorithm is.

To take the randomness of the algorithm into account, we use certain statistics and plot whisker-boxes to compare the values:

- μ - mean of the number of iterations as well as error
- σ - standard deviation of both components
- 5th, 50th (median) and 95th percentiles of the measurements

In order to assess the method, we evaluate obtained results against the results obtained by Random Search Space Sampling for Deterministic Gradient-Descent Algorithm without adding Gaussian Processes. The statistics for summing up results for both algorithms are also the same.

3.13 Results

3.13.1 Results for Pinter's Function

The first test function is Pinter's Function $f(x)$ for multi-dimensional vector x, where x^* is the desired global minimizer, is:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1 P_1(x)) + \sin^2(g_2 P_2(x)) \tag{3.29}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{3.30}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{3.31}$$

The values of the variables are:

$$s = 0.05; \tag{3.32}$$

$$g_1 = 20; \tag{3.33}$$

$$g_2 = 10; \tag{3.34}$$

The maximal number of iterations is $n = 500$. Every iteration we are going to randomly pick and rank $k = 100$ starting points and chose $p = 20$ best of them. The domain of the algorithm is going to be 2-dimensional, that is $f : \mathcal{R}^2 \to \mathcal{R}$ and it is going to contained in:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.35}$$

$$0 \leq x_1 \leq 1000 \tag{3.36}$$

$$0 \leq x_2 \leq 1000 \tag{3.37}$$

The results below shows the number of function evaluations (iterations) for the case where the domain is between 0 and 10^x for each dimension. The graph below shows the number of function evaluations in the 10-logarithmic scale it takes for the algorithm to reach the optimal solution within the accuracy error 1:

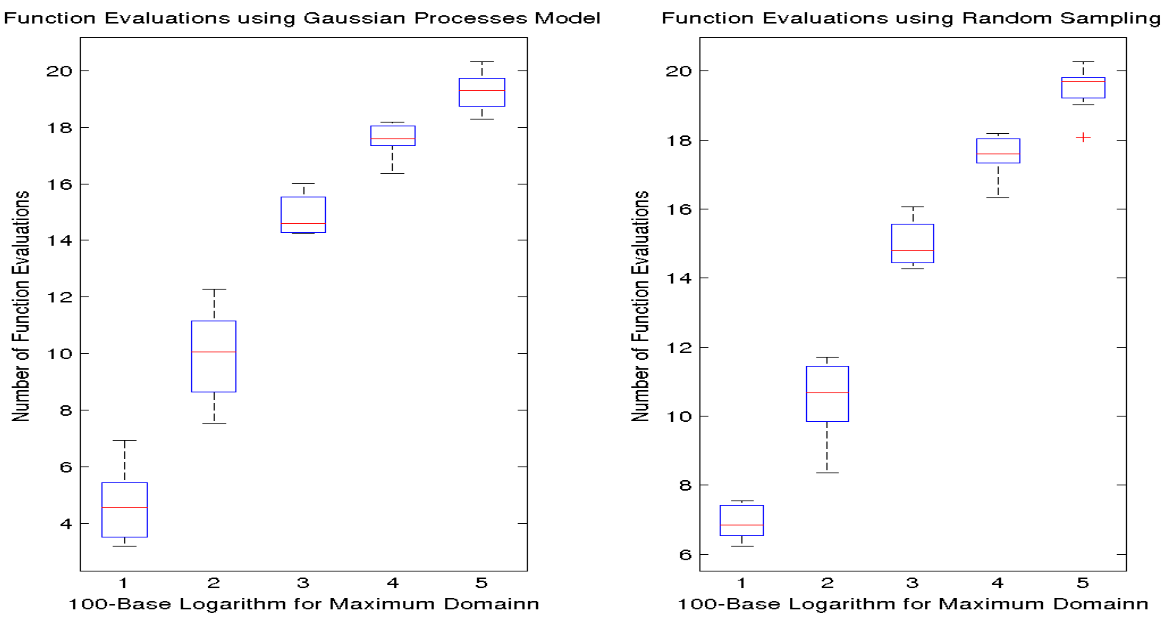

FIGURE 3.5: Number of Function Evaluations for Pinter Function - GP Results First Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 1:

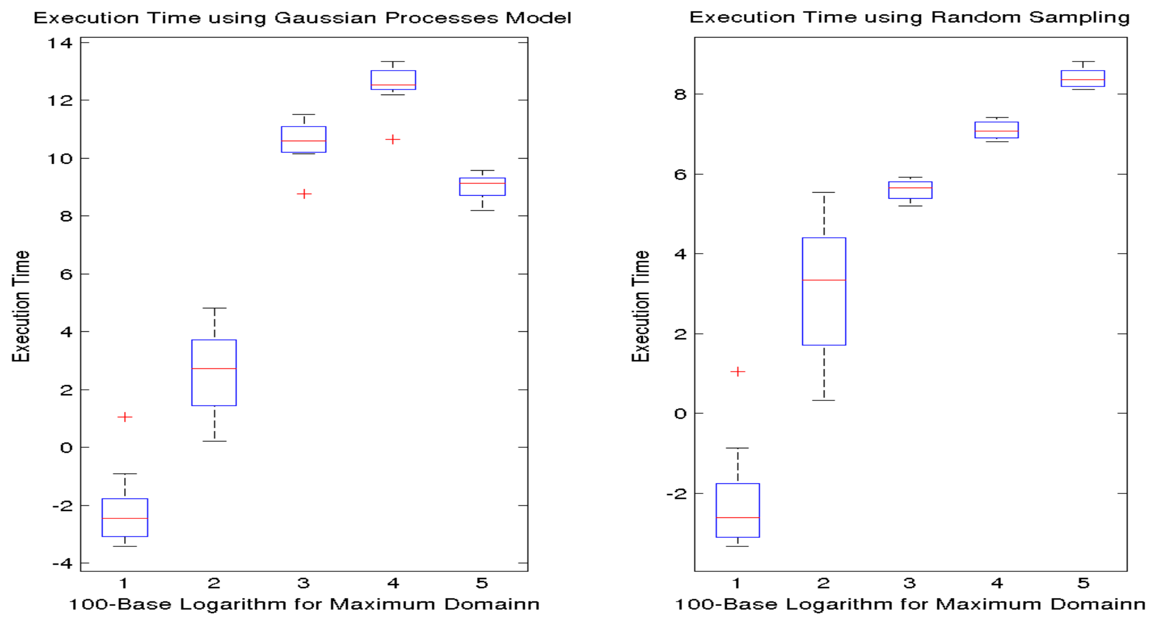

FIGURE 3.6: Execution Time for Pinter Function - GP Results First Barrier

The graph below shows the number of function evaluations in the 10-logarithmic scale it takes for the algorithm to reach the optimal solution within the accuracy error 10:

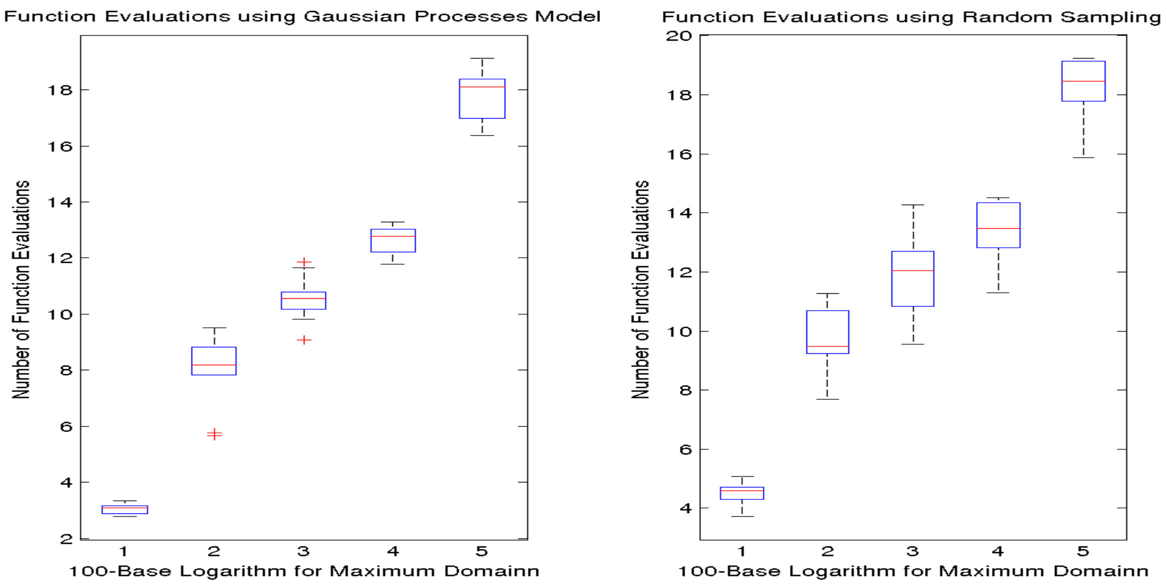

FIGURE 3.7: Number of Function Evaluations for Pinter Function - GP Results Second Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 10:

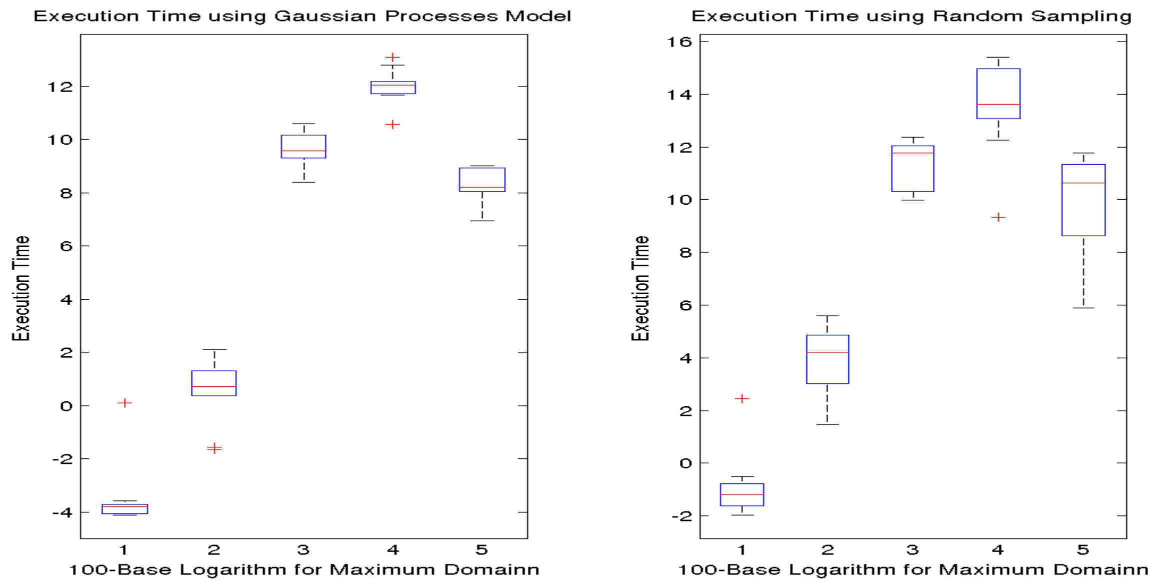

FIGURE 3.8: Execution Time for Pinter Function - GP Results Second Barrier

3.13.2 Results for Michalewicz Function

Another test function we are going to use to evaluate our method is Michalewicz Function. For n-dimensional case, it is of the following form:

$$f(x) = \sum_{j=1}^{n} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{3.38}$$

The results below shows the number of function evaluations (iterations) for the case where the domain is between 0 and 10^x for each dimension. The graph below shows the number of function evaluations it takes for the algorithm to reach the optimal solution within the accuracy error 0.1:

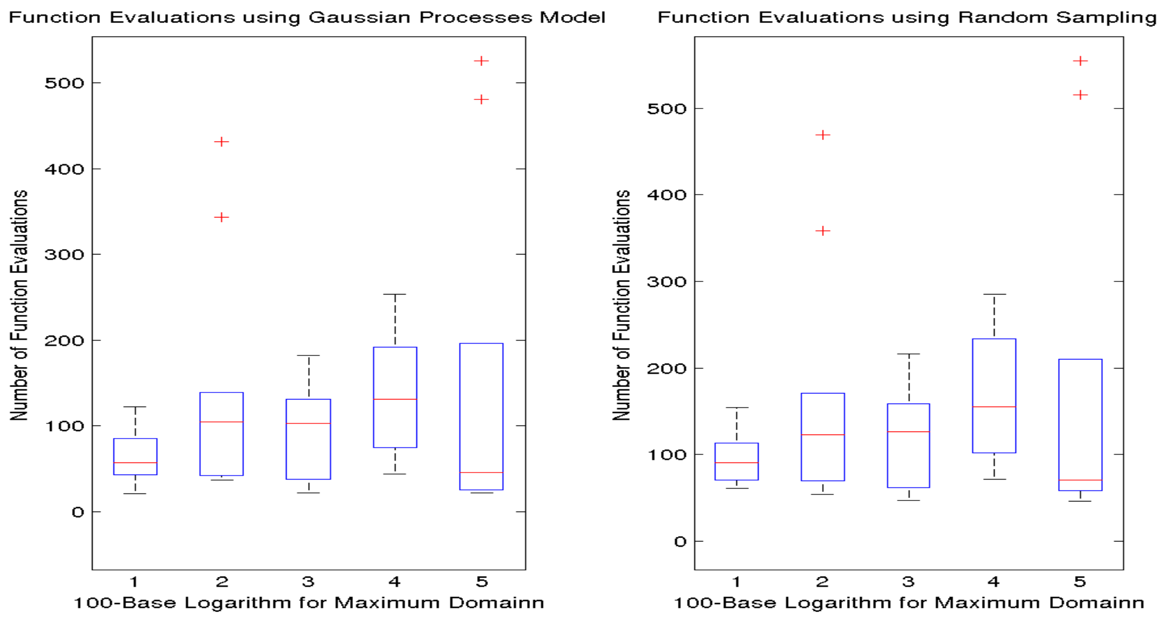

FIGURE 3.9: Number of Function Evaluations for Michalewicz Function - GP Results First Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 0.1:

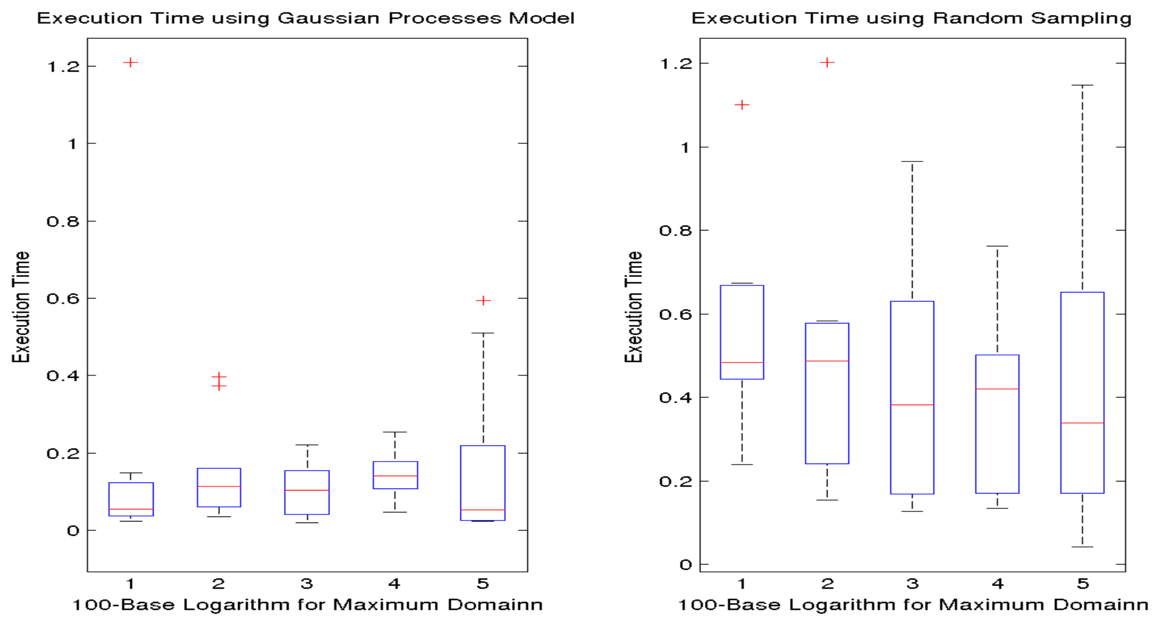

FIGURE 3.10: Execution Time for Michalewicz Function - GP Results First Barrier

The graph below shows the number of function evaluations it takes for the algorithm to reach the optimal solution within the accuracy error 1:

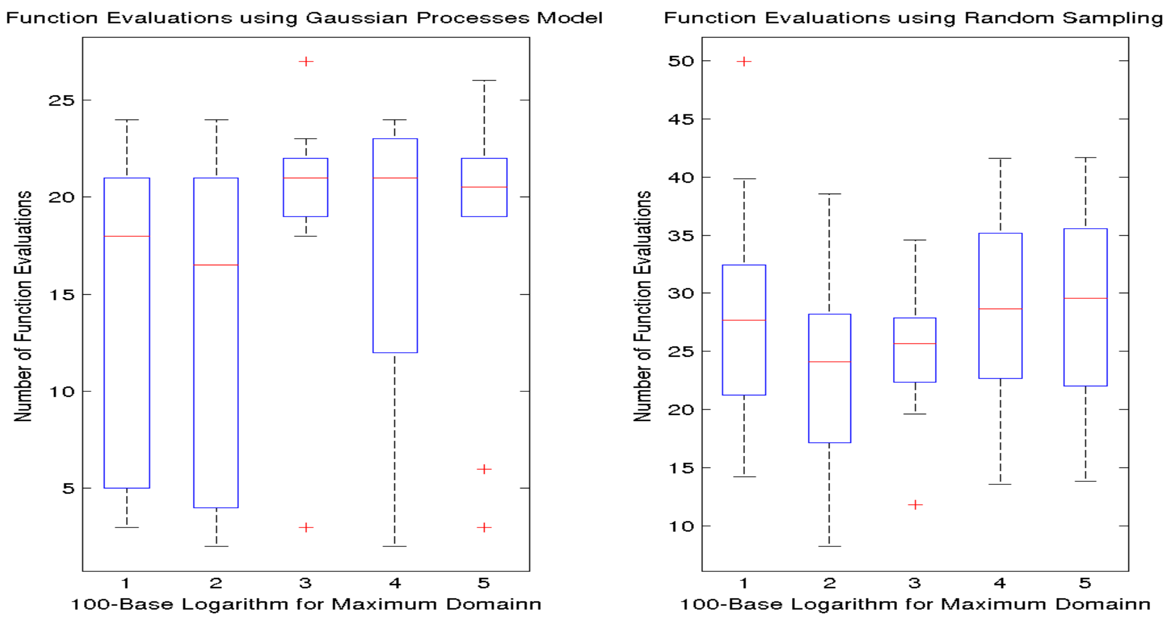

FIGURE 3.11: Number of Function Evaluations for Michalewicz Function - GP Results Second Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 1:

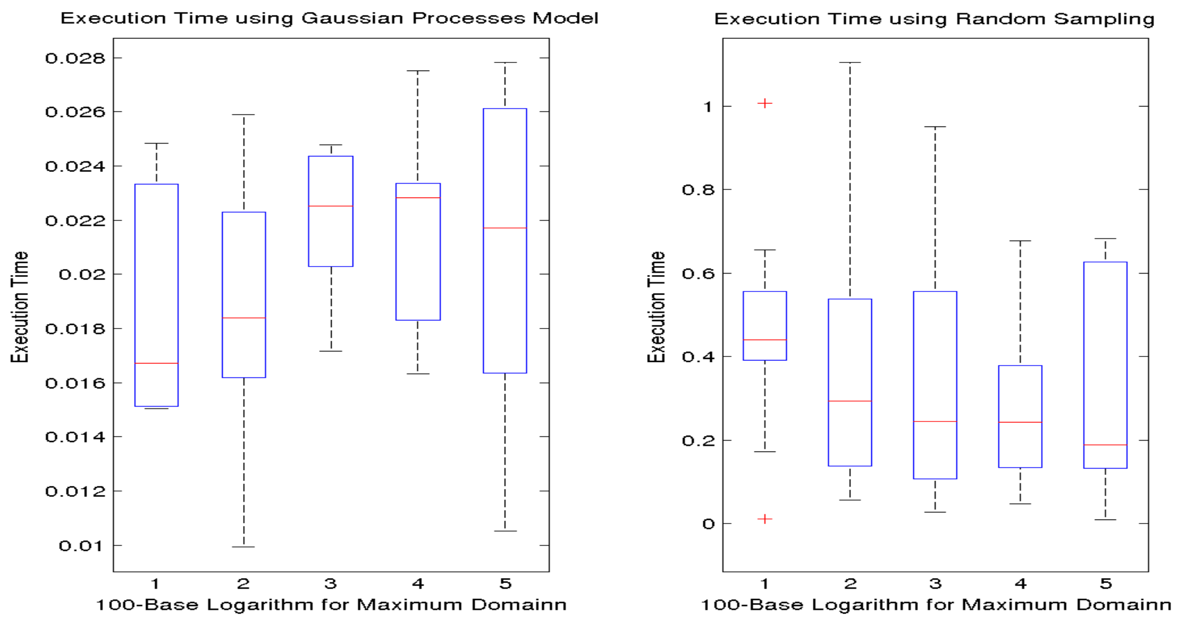

FIGURE 3.12: Execution Time for Michalewicz Function - GP Results Second Barrier

3.13.3 Results for Rastrigin Function

The last function used for evaluation is Rastrigin Function:

$$f(x) = An + \sum_{j=1}^{n}(x_i^2 - A\cos(2\pi x_i)) \tag{3.39}$$

Its gradient is:

$$\frac{\delta f(x)}{\delta x_i} = 2x_i + 2A\pi\sin(2\pi x_i)) \tag{3.40}$$

We are testing the function for $A = 10$.

The domain is enclosed between 0 and 100^x for each dimension.

The graphs show the number of function evaluations (iterations) for the case where the domain is between 0 and 100^x for each dimension. The graph below shows the number of function evaluations it takes for the algorithm to reach the optimal solution within the accuracy error 0.1:

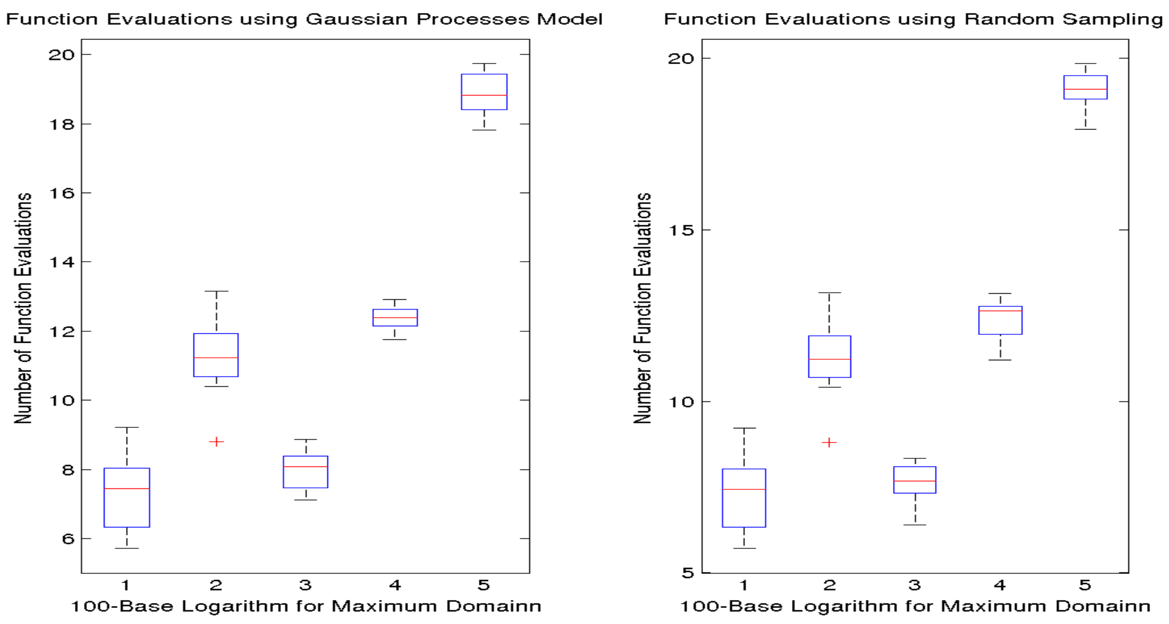

FIGURE 3.13: Number of Function Evaluations for Rastrigin Function - GP Results First Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 0.1:

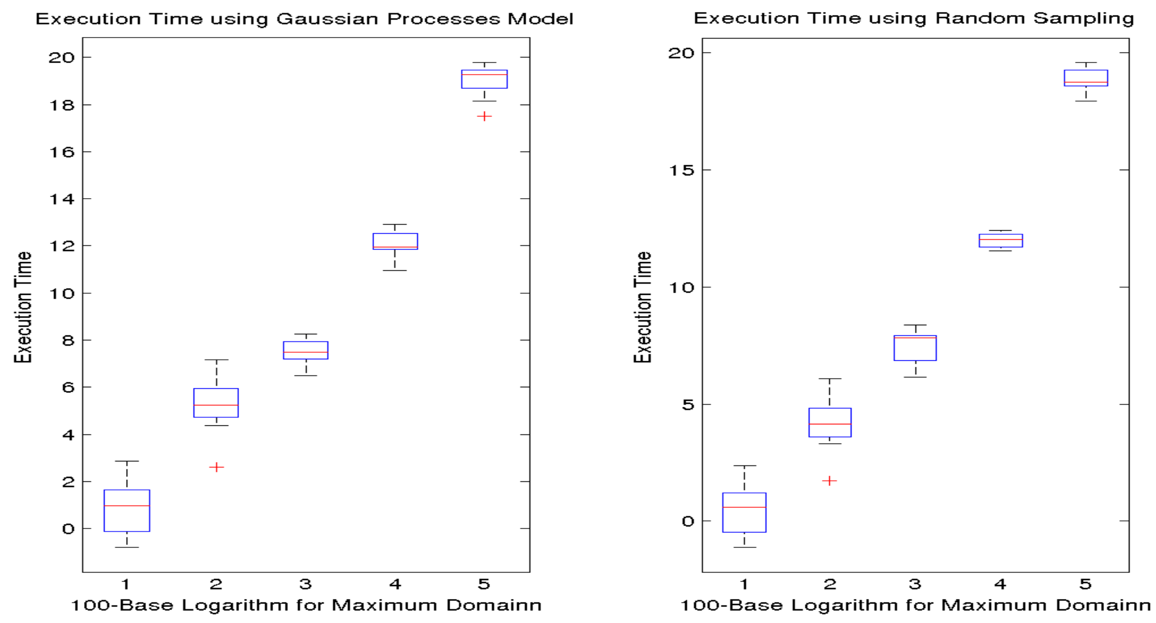

FIGURE 3.14: Number of Function Evaluations for Rastrigin Function - GP Results First Barrier

The results below shows the number of function evaluations (iterations) for the case where the domain is between 0 and 100^x for each dimension. The graph below shows the number of function evaluations it takes for the algorithm to reach the optimal solution within the accuracy error 1:

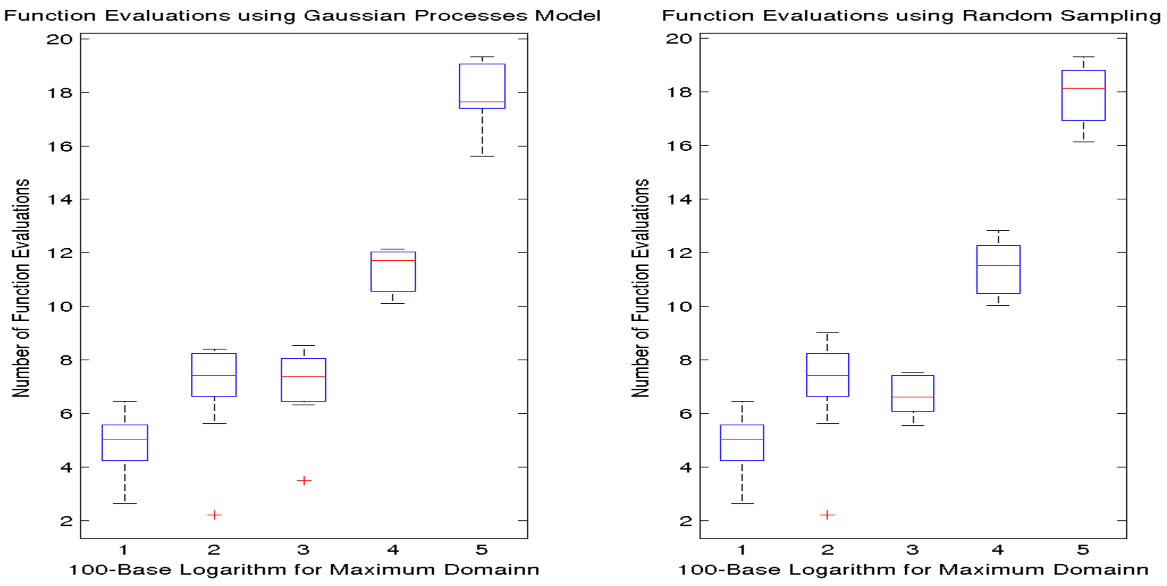

FIGURE 3.15: Number of Function Evaluations for Rastrigin Function - GP Results Second Barrier

The graph below shows how long it takes for the algorithm to reach the optimal solution within the accuracy error 1:

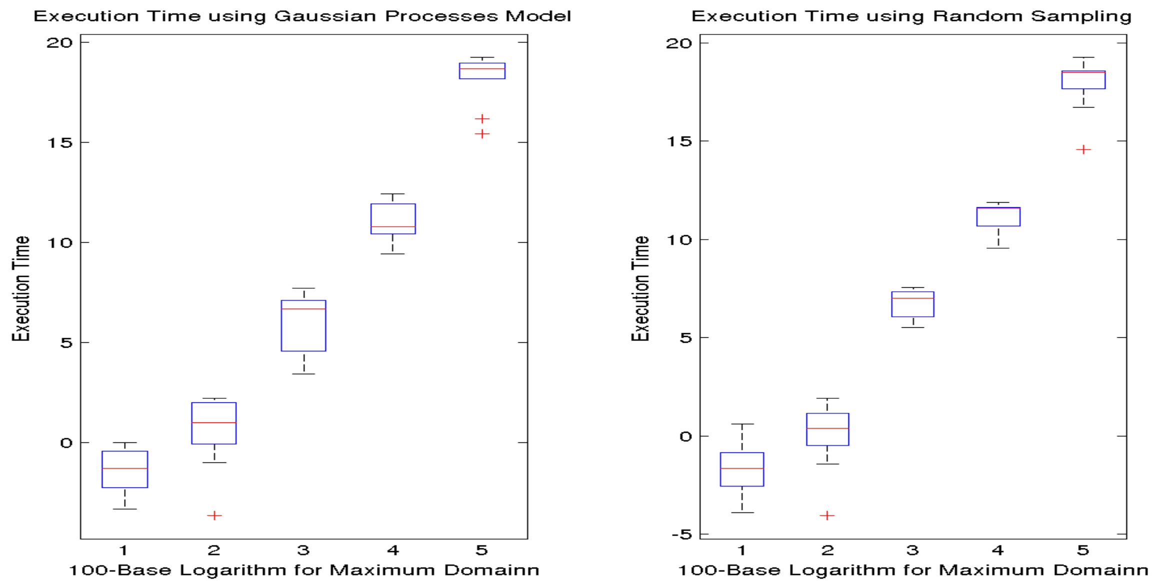

FIGURE 3.16: Number of Function Evaluations for Rastrigin Function - GP Results Second Barrier

3.14 Conclusions

3.14.1 Number of Function Evaluations and Error Accuracy

From the constructed results and graphs plotted for the test functions we conclude that our proposed method performs, in general, better than simply using random sampling in the sense that they decrease the number of function evaluations to reach certain level of error accuracy. However, the level of improvement is not the same across all of the test cases and domain sizes.

In particular, we note that the number of function evaluations needed to reach the error accuracy below 10 in the case of the Pinter function or the error accuracy below 1 in the case of the Michalewicz function is considerably smaller than the results for the corresponding tests in the case of the original Random Sampling of Starting Points. Therefore, we conclude that the method is efficient in the beginning of the search. Afterwards, when looking for the values of accuracy error below the second error accuracy barrier, the performance of the method is still better than that of simple random sampling, but it is not so considerable.

It might be caused by the fact that Gaussian Processes in our model at first helpfully deflect searching through the explored regions. However, due to the extremely big number of local minima in our test functions, our global minimum might be covered by some Gaussian Processes from its neighbor convexity areas which hold local minima and when the algorithm tries to explore the convexity area which contains global minimum, it is deflected by those Gaussian Processes. Another reason for such a trend might be the fact that we use the constant number of Gaussian Processes. Hence, it might turn out that we have too few or too many Gaussian Processes to accurately cover different convexity areas of the explored domain.

3.14.2 Computational Complexity and Execution Time

Another important factor to consider is the complexity of our method and its impact on the general performance of the algorithm. As could be seen in the graphs above, the method does not add much complexity to the proposed algorithm. Because the test functions used are not computationally exhaustive, the contribution of function evaluations to the total execution time is not as significant as the algorithm itself. However, the difference between the proposed algorithm and Random Search is not considerable. One of the reasons for this could be the fact that the most computationally exhaustive part of the algorithm is not ranking the starting point, but running local, deterministic gradient search.

As can be noted in the graph below, the function presented above has a huge number of local minima.

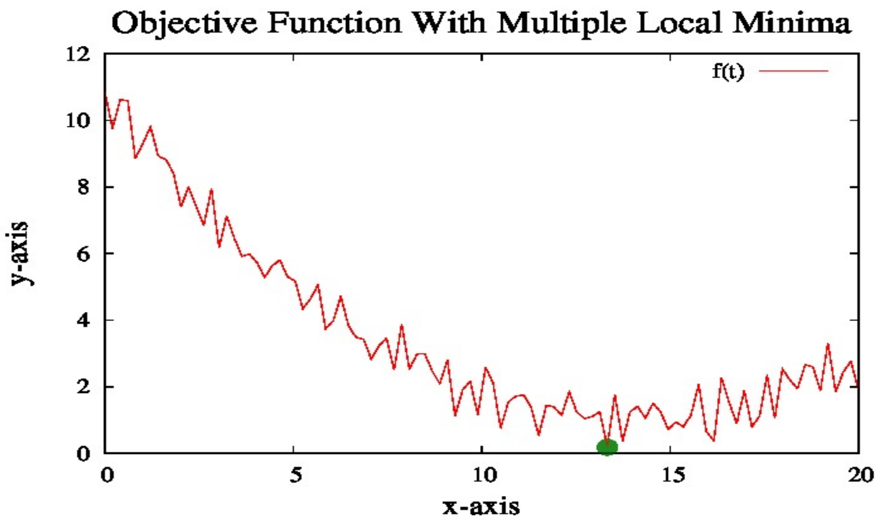

FIGURE 3.17: Function graph

Such a situation is problematic for our method due to the fact that maintaining, storing and utilizing our model of Gaussian Processes would require huge amount of space and computational power. Also, it might turn out that sometimes the evaluation of the model itself takes longer than many evaluations of objective function. In such cases, our method would perform worse than simple random sampling although the number of function evaluations could be smaller.

3.14.3 Final Conclusion and Method Weaknesses

The proposed method performs better for the test functions than Random Sampling of Starting Points. Another advantage of our methods is the fact that there are many parameters which could be used to tune it to specific problems, for example, it could be experimented what is the best method of merging Gaussian Processes or the optimal weight below which Gaussian Processes should be discarded from our model. Interestingly, we also notice that the bigger error accuracy can be accepted, the better the proposed method performs in comparison to Random Sampling of Starting Points. As explained, the error accuracy beyond which the proposed method loses that advantage might be caused by the number of factors such as, for example, number of Gaussian Processes or number of local minima that the objective function has.

However, we found certain weaknesses of our method. The most severe one of them is the fact that it might scale badly whenever we have a problem with the huger number of local minima. In such a case, our method might become intractable and computationally too challenging and would not be advisable. However, in such a case we can tune parameter to fit the problem by, for example, setting the maximum number of Gaussian Processes to certain, acceptable value. Another weakness of the proposed method is certainly the huge number of parameters and it might be challenging to create certain heuristics or methods of choosing values for them. Despite those drawbacks, the methods and the approach itself of using Gaussian Processes for global optimization might be considered successful and promising for future works.

Chapter 4

Adding Stochastic Term to Hyperbolic Cross Points

4.1 Properties of Objective Function

Hyperbolic Cross Points method, described and analyzed by E. Novak in [2], is used for finding a global minimum of function $f(x) : \mathcal{R}^n \to \mathcal{R}$ for box constraints, that is every constraint for dimension i is expressed as an interval:

$$a_i \leq x_i \leq b_i \tag{4.1}$$

The only assumption is that the following should contain an open set for ϵ:

$$\{x_i \in [a_i, b_i] | \inf_{x_i \in [a_i, b_i]} f(x) + \epsilon\} \tag{4.2}$$

The property above is true for any continuous or upper semi-continuous function.

4.2 Hyperbolic Cross Points

Now, we explain how hyperbolic cross points are generated. To show the process, we use the example where the constraints are given as:

$$[a, b] = \prod_{i=1}^{d} [-0.5, 0.5] \tag{4.3}$$

The example is based on work in [12]. Hyperbolic cross point is defined as any point which have co-ordinates equal to:

$$x_i = \pm \sum_{j=1}^{k} a_j 2^{-j} \ \ , a_j \in \{0, 1\} \tag{4.4}$$

Another very important notion used in producing hyperbolic cross points is the level of a point. It is defined as the sum of its components:

$$lev(x) = \sum_{i=1}^{d} lev(x_i) \tag{4.5}$$

The starting point with of level $lev(x_0) = 0$ is a zero point. The neighbor of hyperbolic cross point x is its neighbor which is different only in one coordinate. The neighbor point of x of level k is the neighbor of degree m which has level equal to $k + m$ and is defined as:

$$x'_i = x_i \pm 2^{-lev(x_i)-m} \wedge x'_j = x_j \text{ for } j \neq i \tag{4.6}$$

To explain how the point are created more clearly, we show it by the following example.

Let us assume that x is:

$$x = (0.5, 0) \tag{4.7}$$

The level of x is 1. The neighbor of x of degree 1 can be:

$$x' = (0.5, 0.5) \tag{4.8}$$

or:

$$x' = (0.5, -0.5) \tag{4.9}$$

4.3 Mapping Hyperbolic Cross Points to Other Constraints

The definition of hyperbolic cross points given above are given for the constraints of the form $[-0, 5, 0.5]$ for each coordinate. For the general case where the coordinates are defined as:

$$[a, b] = \prod_{i=1}^{d} [a_i, b_i] \tag{4.10}$$

where $a_i \in \mathcal{R}$ and $b_i \in \mathcal{R}$. The hyperbolic cross point is now defined as:

$$x_i = \pm \frac{b_i - a_i}{2} \sum_{j=1}^{k} a_j 2^{-j} \ \ , a_j \in \{0, 1\} \tag{4.11}$$

In such a case, the coordinates of starting point is defined as:

$$x_i = \frac{b_i - a_i}{2} \tag{4.12}$$

The neighbors of point x are also defined a little bit different:

$$x'_i = x_i \pm \frac{b_i - a_i}{2} 2^{-lev(x_i)-m} \wedge x'_j = x_j \text{ for } j \neq i \tag{4.13}$$

As can be noticed, the hyperbolic point cross for the general case is just the linear mapping of the base case.

4.4 Graphical Examples of Hyperbolic Cross Points

The following diagrams show the points which have been generated using the hyperbolic cross points method for different levels. The first diagram has the red square marking the initial point in the middle of the domain:

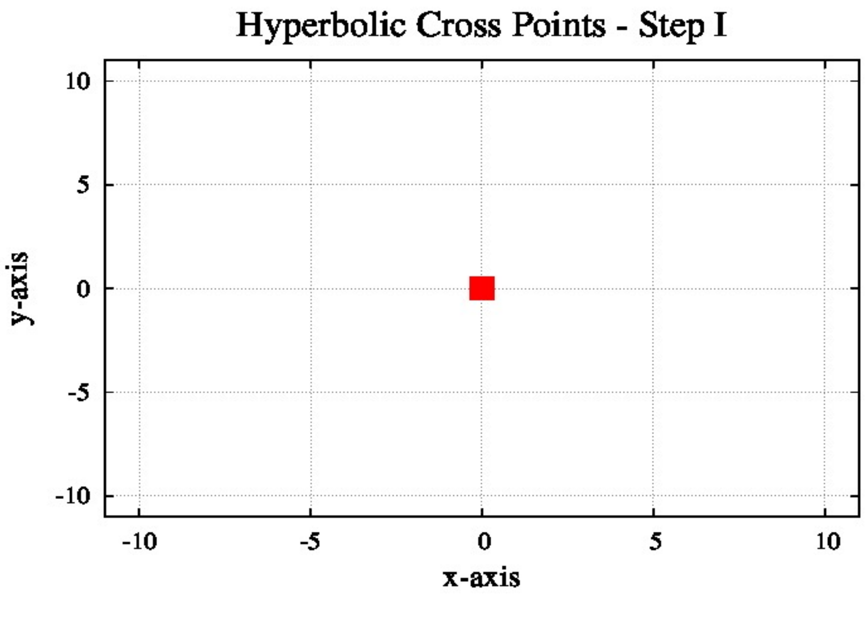

Figure 4.1: Hyperbolic Cross Points Grid - Step I

The next diagram has the green squares added the points which are the neighbors of the initial point (red square):

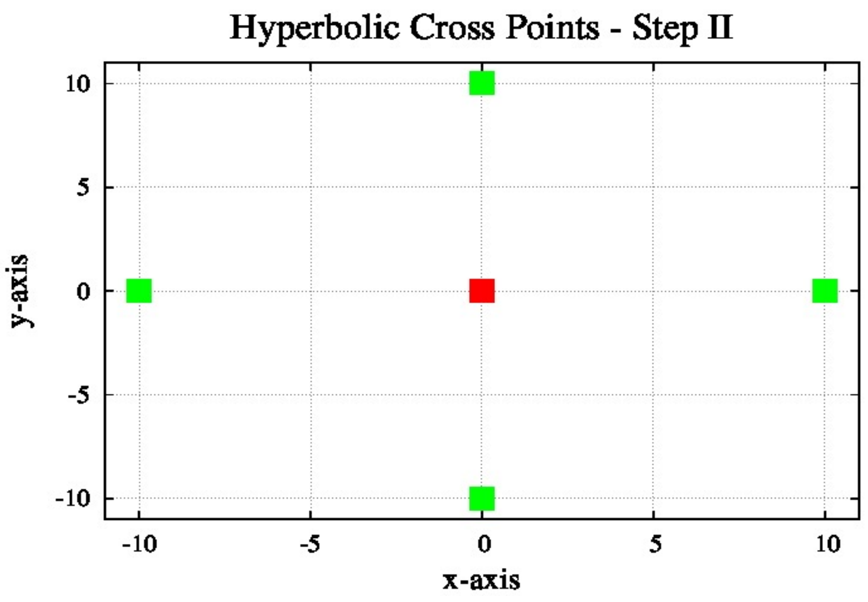

FIGURE 4.2: Hyperbolic Cross Points Grid - Step II

The next diagram has the blue squares added the points which are the neighbors of the green squares (Hyperbolic Cross Points - Level I):

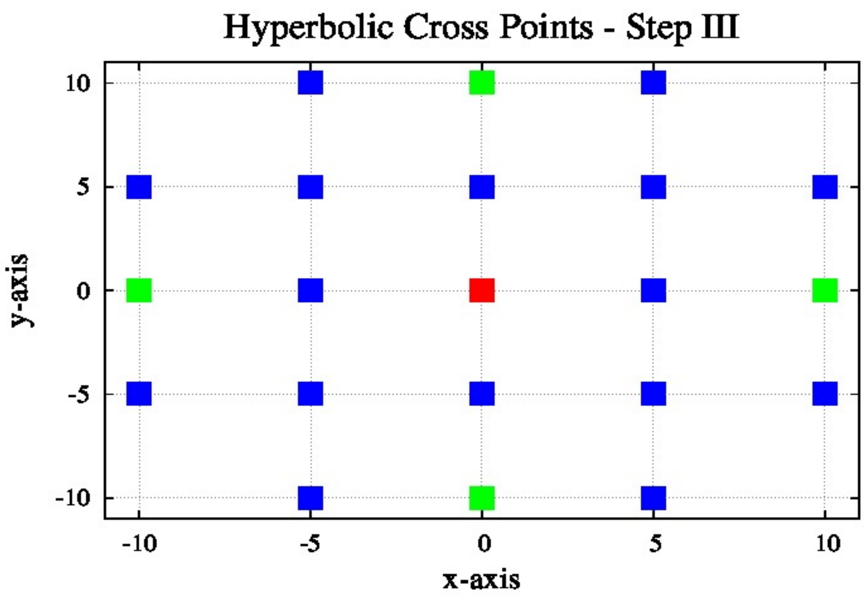

FIGURE 4.3: Hyperbolic Cross Points Grid - Step III

4.5 HPC points in Global Optimization

To use Hyperbolic Cross Points for Global Optimization, we iteratively generate hyperbolic cross points. At each generation, we evaluate our objective function $f(x)$. The final result of using Hyperbolic Cross Points method to generate the set $\mathcal{H}$ of the HPC points is x defined as:

$$x : x \in \mathcal{H} \land \forall x^* \in \mathcal{H} : f(x) \leq f(x^*) \tag{4.14}$$

In practice, at each stage we generate a few candidates for the next point to evaluate and rank them. Then, the best point is picked and the algorithm continues.

The algorithm terminates when we exceed the maximum number of function evaluations or when we reach satisfactory results.

4.6 Ranking Hyperbolic Cross Points

There are two main different methods of finding the next point from candidate neighbors points to be evaluated. Those rankings are adaptive or non-adaptive. They are described and analyzed in [12].

4.6.1 Non-Adaptive Approach

In non-adaptive version of the algorithm, the ranking is used to measure how thoroughly the region has been already searched. To calculate the ranking of a given point, we use a few metrics. First of all, the level of the point is used: the smaller the level of the point is, the less explored the region is. In addition to this, we also take into account the degree of the point. The degree of the point is defined as the number of times it has been chosen as the best point plus one. Similarly to the level, it is also used to express how well explored the search region is and how likely it is to add new information. The ranking is defined quantitatively as:

$$g(x) = lev(x) + deg(x) \tag{4.15}$$

The evaluated value of $g(x)$ is used to order and compare points. The smaller the value of $g(x)$ is, the more preferable point is. Such an approach is called non-adaptive, because it does not depend on which function is optimized.

4.6.2 Adaptive Approach

In adaptive version of the algorithm, we try to add some information that could be used to take advantage of some properties of an objective function. In addition to calculating the level and degree of evaluated point, we also define the following ranking for adaptive version of the algorithm:

$$rank(x) = \#\{y \in Y | y \leq f(x)\} \tag{4.16}$$

The set Y is the set of values evaluated by functions, that is:

$$Y = \{f(x_1), f(x_2), ..., f(x_k)\} \tag{4.17}$$

In other words, the function $rank(x)$ returns what is the ranking of the function value evaluated at point x.

Now, the ranking function $g(x)$ is defined for adaptive version of the algorithm as:

$$g(x) = (lev(x) + deg(x))^{\alpha} rank(x)^{1-\alpha} \text{ where: } \alpha \in [0, 1] \tag{4.18}$$

Depending on the value of α, the algorithm can be set to more or less adaptive. For $\alpha = 0$ the algorithm becomes non-adaptive, and for $\alpha = 1$ it depends only on the function value.

4.6.3 Determinism of HPC Point Ranking

In the current setting of the algorithm, the points are ranked and ordered using the value of $g(x)$. The exact form of $g(x)$ depends on how adaptive our algorithm is expected to be. In the next steps, the best points are evaluated. Depending on whether or now we want the algorithm to continue its execution, it might continue or terminate.

As can be noticed, the given approach makes the algorithm deterministic. In the case of non-adaptive algorithm, we always choose the points in the same order for given constraints independent of the objective function. For non-adaptive approach, the function will be optimized to the same value every time the algorithm is used on it.

Such determinism limits us in a few ways. In particular, we might want to perform Monte Carlo or batch simulations, but running the algorithm many times would not produce different results. Another important fact is that it would not make any sense to run the algorithm in parallel or in distributed systems as they would not produce different results given the same, constant maximum number of evaluations. Hence, to tackle those limitations, we propose adding stochastic term to the method and compare the results.

4.6.4 Adding Stochastic Term to HPC Point Ranking

As has been defined previously, in the deterministic version of the algorithm, the points to be evaluated next are ranked in the following way:

$$EvaluationRank(x) = \#\{y \in Y | y \leq g(x)\} + 1 \tag{4.19}$$

For Non-Adaptive Version, we have:

$$g(x) = lev(x) + deg(x) \tag{4.20}$$

For Adaptive Version, we have:

$$g(x) = (lev(x) + deg(x))^{\alpha} rank(x)^{1-\alpha} \text{ where: } \alpha \in [0,1] \tag{4.21}$$

where:

$$rank(x) = \#\{y \in Y | y \leq f(x)\} \tag{4.22}$$

The proposed change of the algorithm involves adding randomness to the process of ranking.

Currently, the point ranked to be number 1 has the probability of being chosen equal to 1, while all the others have zero probability of being chosen. To change that, we assign each point its own non-zero probability.

We start from selecting n points to be randomly chosen. The first step is to change how points are exactly evaluated. As can be seen in the previous equation, the points can have the same values. In such a case when the points have equal ranking position, we adjust their rankings by given the priority to the points which were evaluated first. Now, having made sure that each point has its unique position in ranking, we can evaluate their position k in the ranking. Each point is going to be assigned specific bounds. The upper value of the bound for kth point is denoted as $B_{up}(k)$ and the lower value of the bound is $B_{down}(k)$. Then, the next step is to draw a number p from a uniform distribution over $[0,1]$. Having that value, we can locate in which band it is, that is for which point the following is true:

$$B_{down}(k) \leq p < B_{up}(k) \tag{4.23}$$

The point for which the equation above is true is chosen for the next evaluation.

4.7 Bounds Calculation

In this section, we present two proposed approaches of calculating probability bounds.

4.7.1 First Approach - Halving Total Probabilities

The first presented approach assumes that the following equation should be preserved:

$$p(x_i) = 2\sum_{j=i+1}^{n} p(x_j) \tag{4.24}$$

In words, this approach expresses the fact choosing a point ranked k is twice as likely as choosing any point ranked higher k.

The value of the upper bound is defined as:

$$B_{up}(x_k) = \frac{1}{3^{(k-1)}} \tag{4.25}$$

The value of the lower bound is defined as:

$$B_{down}(x_k) = \frac{1}{3^k} \tag{4.26}$$

As can be noticed, the upper value of the first point is equal to:

$$B_{up}(1) = \frac{1}{3^{(k-1)}} = 1 \tag{4.27}$$

In addition to this, we can notice that:

$$B_{upper}(k) = B_{down}(k+1) \tag{4.28}$$

And, finally, the lower bound of the last point is explicitly defined to be:

$$B_{down}(x_n) = 0 \tag{4.29}$$

Hence, because $B_{up}(1) = 1$, $B_{down}(n) = 0$ and $B_{down}(k) = B_{up}(k+1)$, we can be sure that the whole domain $[0, 1]$, which we draw a random point from, is covered by the bound assignment.

We show that the bounds given above satisfies our assumption:

$$\frac{p(x_j)}{\sum_{j=i}^{n} p(x_j)} = \frac{B_{up} - B_{down}}{\sum_{j=i}^{n} p(x_j)} = \frac{\frac{1}{3^0} - \frac{1}{3}}{\sum_{j=i}^{n} p(x_j)} = \frac{\frac{1}{3}}{\sum_{j=i}^{n} p(x_j)} \tag{4.30}$$

Probability of choosing a point which is after k, given than the point equal to or after is chosen, is given by:

$$\frac{p(x_i)}{\sum_{j=i+1}^{n} p(x_j)} = \frac{1-p(x_i)}{\sum_{j=i}^{n} p(x_j)} = \frac{\frac{2}{3}}{\frac{1}{3}} = 2 \tag{4.31}$$

Hence, our assumption is satisfied. However, it is important to notice to make sure that we cover the whole region $[0,1]$ over which we sample our random variable we did specify that $B_{down}(n) = 0$, the probability of the last point to be chosen is, in fact, equal to:

$$p(x_n) = 1 - \sum_{i=1}^{n} (B_{up}(i) - B_{down}(i)) \tag{4.32}$$

It results from this that our assumption does not hold for the following case:

$$p(x_{n-1}) = 2p(x_n) \tag{4.33}$$

In other words, we add the residual probability to the last term.

4.7.1.1 Example of Random HPC Point Ranking for Halving Total Probabilities

Let us assume that $n = 3$, that is there are four points to be ranked. Then, each point has the following bounds:

- x_1: $B_{up} = 1$, $B_{down} = \frac{1}{3}$
- x_2: $B_{up} = \frac{1}{3}$, $B_{down} = \frac{1}{9}$
- x_3: $B_{up} = \frac{1}{9}$, $B_{down} = 0$

Also:

$$p(x_1) = \frac{2}{3} \tag{4.34}$$

and:

$$2(p(x_2) + p(x_3) = 2(\frac{2}{9} + \frac{1}{9} = 2\frac{1}{3} = \frac{2}{3} \tag{4.35}$$

Now, if we happen to draw a random number $p = 0.5$, then we would chose the point x_1, because:

$$B_{down}(x_1) < 0.5 \leq B_{up}(x_1) \rightarrow \frac{1}{3} < 0.5 \leq 1 \tag{4.36}$$

4.7.2 Second Approach - Halving Consecutive Probabilities

The second presented approach assumes that the following equation should be preserved true:

$$p(x_i) = 2p(x_{i+1}) \tag{4.37}$$

Hence, the upper bound is equal to:

$$B_{up}(x_k) = \frac{1}{2^{k-1}} \tag{4.38}$$

and its lower bond is:

$$B_{down}(x_k) = \frac{1}{2^k} \tag{4.39}$$

and:

$$p(x_k) = B_{up} - B_{down} = \frac{1}{2^{k-1}} - \frac{1}{2^k} = \frac{1}{2^k} \tag{4.40}$$

Hence, we can prove our assumption:

$$\frac{p(x_k)}{p(x_{k+1})} = \frac{\frac{1}{2^k}}{\frac{1}{2^{k+1}}} = \frac{1}{\frac{1}{2}} = 2 \tag{4.41}$$

Similarly to the previous method, we can define see that:

$$B_{up}(1) = 1 \tag{4.42}$$

and:

$$B_{down}(k) = B_{up}(k+1) \tag{4.43}$$

To make sure that the whole sample domain $[0, 1]$, we write that:

$$B_{down}(n) = 0 \tag{4.44}$$

4.7.2.1 Example of Random HPC Point Ranking for Halving Consecutive Probabilities

Let us assume that $n = 3$, that is there are four points to be ranked. Then, each point has the following bounds:

- x_1: $B_{up} = 1$, $B_{down} = \frac{1}{2}$
- x_2: $B_{up} = \frac{1}{2}$, $B_{down} = \frac{1}{4}$

- x_3: $B_{up} = \frac{1}{4}$, $B_{down} = 0$

Also:

$$p(x_1) = \frac{1}{2} \wedge p(x_2) = \frac{1}{4} \tag{4.45}$$

and:

$$\frac{p(x_1)}{p(x_2)} = \frac{\frac{1}{2}}{\frac{1}{4}} = 2 \tag{4.46}$$

Now, if we happen to draw a random number $p = 0.3$, then we would chose the point x_2, because:

$$B_{down}(x_2) < 0.3 \leq B_{up}(x_2) \rightarrow 0.25 < 0.3 \leq 0.5 \tag{4.47}$$

4.8 Convergence of Randomized HCP Algorithm

For deterministic version of the algorithm (both adaptive and non-adaptive), it is relatively easy to show the convergence of the method. First, we define the notation used. The following sequence shows function evaluations after n steps:

$$N_n(f) = (f(x_1), f(x_2), .., f(x_n)) \tag{4.48}$$

We can also define the inverse mapping, that is:

$$x_k = \psi_k(N_n(f)) \tag{4.49}$$

To denote a global optimization method, we use the following notation:

$$\phi_n = (\psi_1, \psi_2, ..., \psi_n) \tag{4.50}$$

The error of our global optimization method can be now defined as:

$$\Delta(\phi_n, f) = f(x^*) - \inf_{x \in [a,b]} f(x) \tag{4.51}$$

Where x^* denotes our obtained solution, and $\inf_{x\in[a,b]} f(x)$ the true global minimum. In HPC method, it simply takes the minimum of $N_n(f)$, that is:

$$x^* = \min(N_n(f)) = \min(f(x_1), f(x_2), .., f(x_n)) \tag{4.52}$$

Now, to show that the method converges, we need to show that the following statement is true:

$$\lim_{n\to\infty} \Delta(\phi_n, f) = 0 \tag{4.53}$$

It means that after infinite number of iterations, we are guaranteed to find the global minimum of our objective function f. The following equation is true, because Hyperbolic Cross Methods simply tends to imitate dense grid after enough number of iterations.

To show that for our randomization of the method the algorithm is still guaranteed to converge, we can show that every point in the domain of the function is guaranteed to be tested. As can be noticed, the probability of not choosing some points is:

$$\lim_{n \to \infty} (1 - p(\bar{x_i}))^n = 0 \tag{4.54}$$

Hence, we are guaranteed to evaluate every point as it would be evaluated in the dense grid, which does in fact guarantee convergence.

4.9 Stopping Criteria for Test Simulations

For all of the ranking methods used, we assume that if a point has the neighbors closer than $\epsilon = 0.1$ and has the value of objective function lower than all the neighbors, it is assumed to be a local minimizer. What is more, when the level of the point exceeds certain constant for some dimension $lev(x_i) = 10$, we do not proceed with finding its neighbors as the difference is too small, which could result in substantially increasing the complexity and length of the simulations.

4.10 Evaluation Methodology

To test the proposed addition of stochastic term to the algorithm, we measure the performance of the algorithm against four metrics:

1. Number of function evaluations and execution time before the algorithm reaches the optimal solution within the accuracy error of 10 to the true, optimal solution
2. Number of function evaluations and execution time before the algorithm reaches the optimal solution within the accuracy error of 1 to the true, optimal solution

Each of the method is tested over various domains. We are going to plot whisker plots from the obtained results. The algorithms will be tested on three functions described in the background. The performance of the proposed methods is measured against the original version of Stochastic Gradient Descent. In the presented graphs for each function, the following notation is used:

- Non-Adaptive - Hyperbolic Cross Points using Non-Adaptive Ranking, that is choosing the point with the lowest overall level and degree.

- Adaptive - Hyperbolic Cross Points using both the total level and degree of the point as well as its current position in terms of objective function relative to other evaluated points. The value of alpha is kept constant for all the simulations and is equal to: $\alpha = 0.5$.
- Stochastic - Hyperbolic Cross Points using the total level and degree of the point, its current position in terms of objective function relative to other evaluated points as well as stochastic term added to the final ranking. The value of alpha is kept constant for all the simulations and is equal to: $\alpha = 0.5$.

4.11 Results

The number of iterations is $n = 100$. The domain of the algorithm is going to be 2-dimensional, that is $f : \mathcal{R}^2 \to \mathcal{R}$.

4.11.1 Results for Pinter Function

Another test function is Pinter Function $f(x)$ for multi-dimensional vector x, where x^* is the desired global minimizer, is:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1 P_1(x)) + \sin^2(g_2 P_2(x)) \tag{4.55}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{4.56}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{4.57}$$

The values of the variables are:

$$s = 0.05; \tag{4.58}$$

$$g_1 = 20; \tag{4.59}$$

$$g_2 = 10; \tag{4.60}$$

The graph below shows the number of function evaluations (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 10:

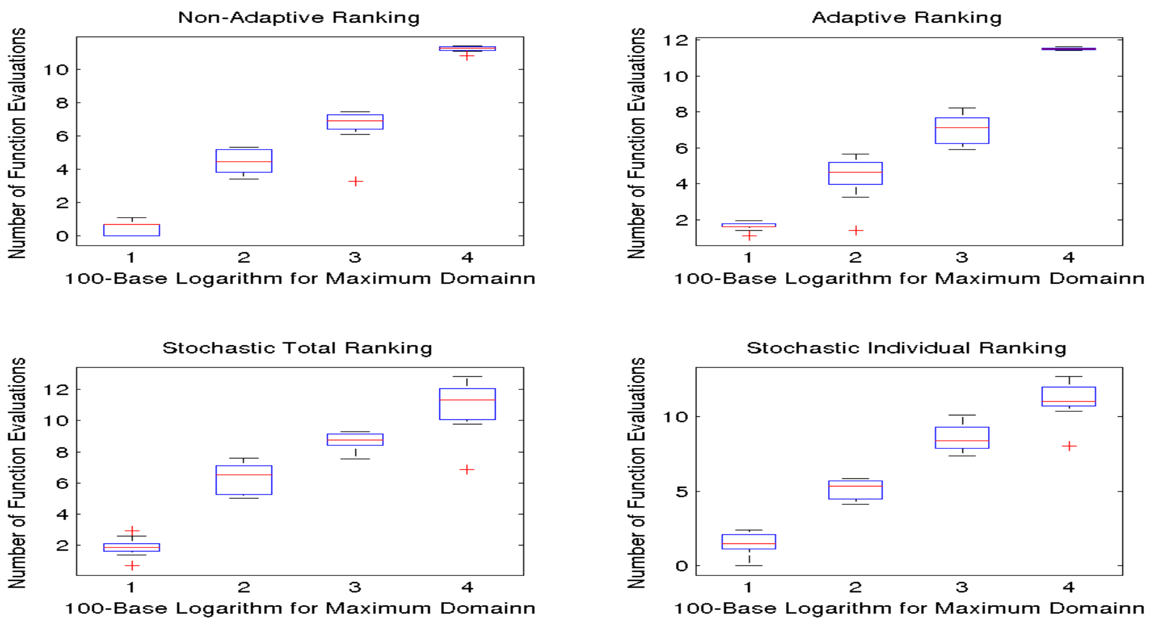

FIGURE 4.4: HCP - Number of Function Evaluations for Pinter Function - First Accuracy Barrier

The graph below shows the execution time (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 10:

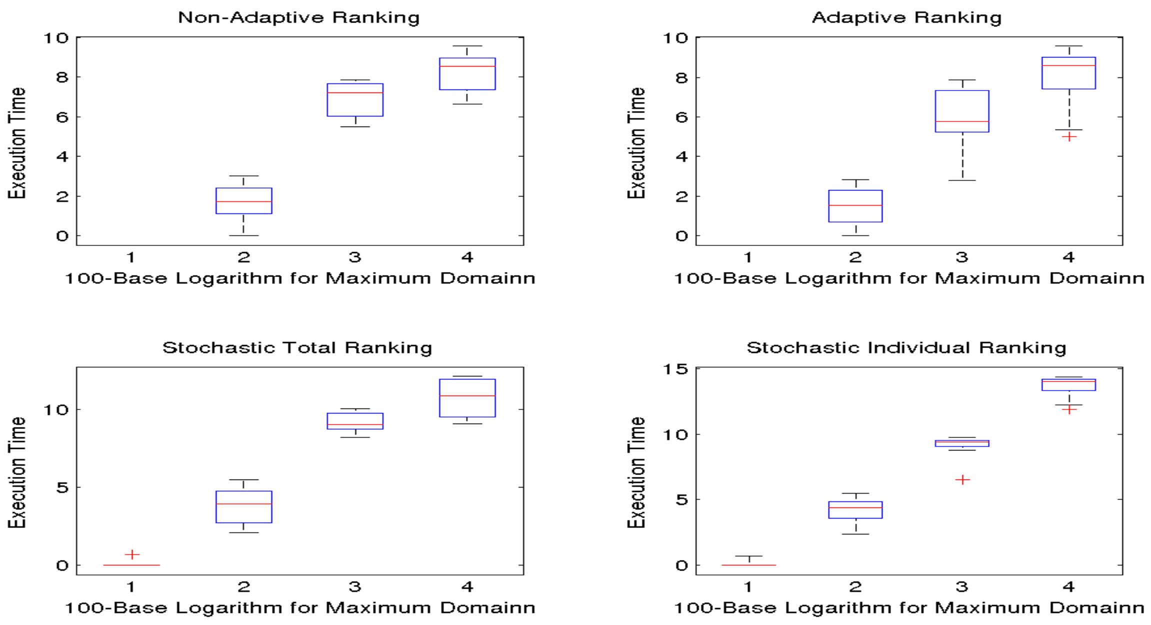

FIGURE 4.5: HCP - Execution Time for Pinter Function - First Accuracy Barrier

The graph below shows the number of function evaluations (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 1:

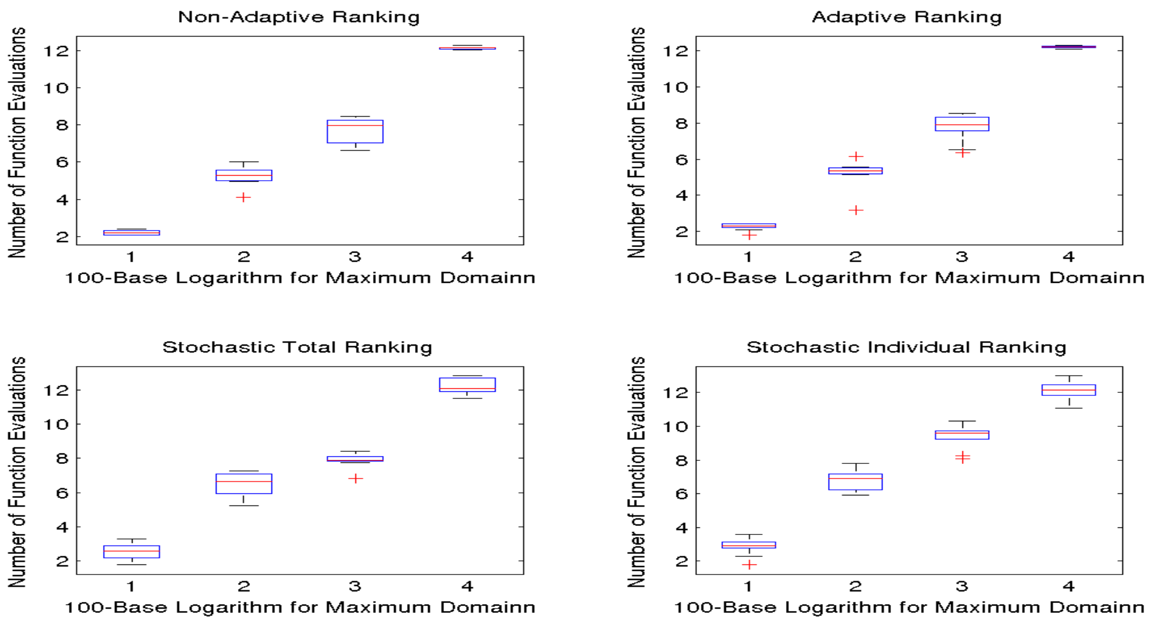

FIGURE 4.6: HCP - Number of Function Evaluations for Pinter Function - Second Accuracy Barrier

The graph below shows the execution time (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 1:

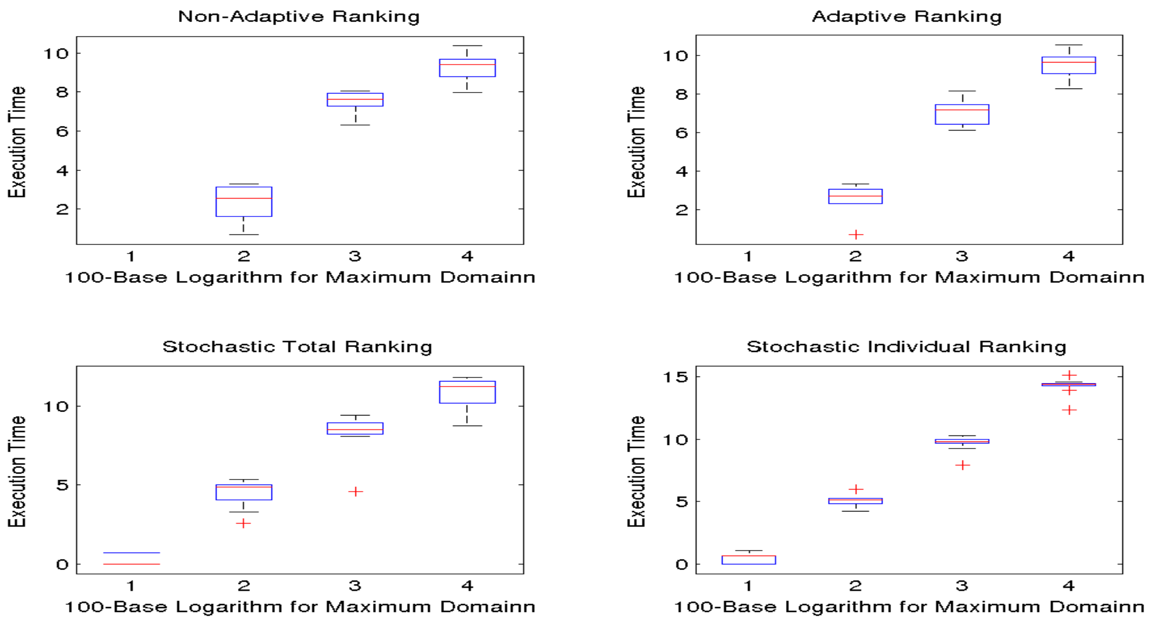

FIGURE 4.7: HCP - Execution Time for Pinter Function - Second Accuracy Barrier

4.11.2 Results for Michalewicz Function

The first test function we are going to use to evaluate our method is Michalewicz Function. For n-dimensional case, it is of the following form:

$$f(x) = \sum_{j=1}^{n} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{4.61}$$

The graph below shows the number of function evaluations for the algorithm to reach the optimal solution within the accuracy error of 10:

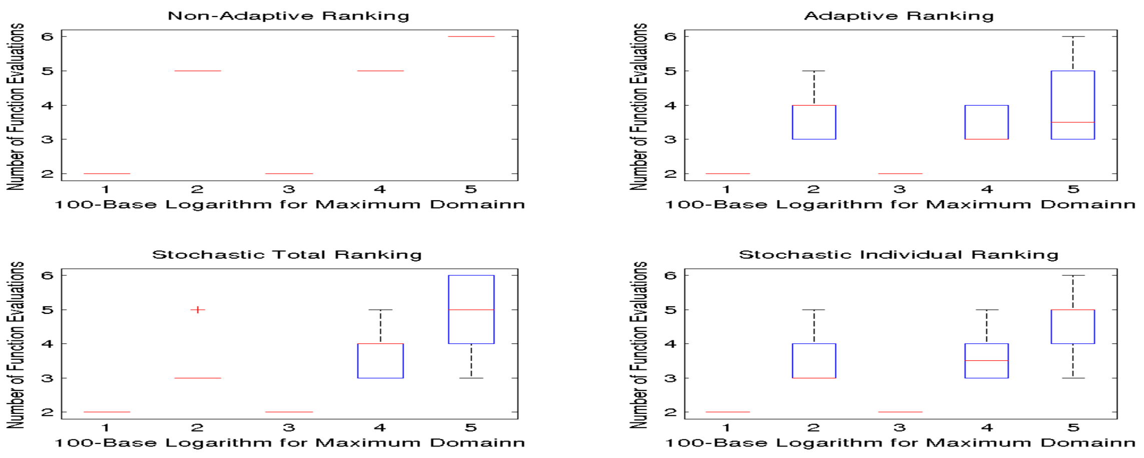

FIGURE 4.8: HCP - Number of Function Evaluations for Michalewicz Function - First Accuracy Barrier

The graph below shows the execution time for the algorithm to reach the optimal solution within the accuracy error of 10:

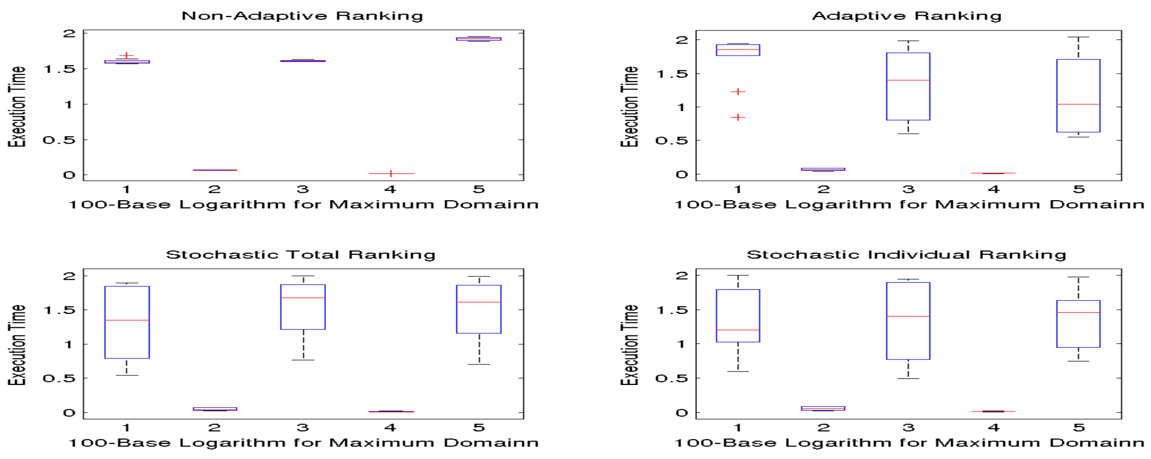

FIGURE 4.9: HCP - Execution Time for Michalewicz Function - First Accuracy Barrier

The graph below shows the number of function evaluations (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 1:

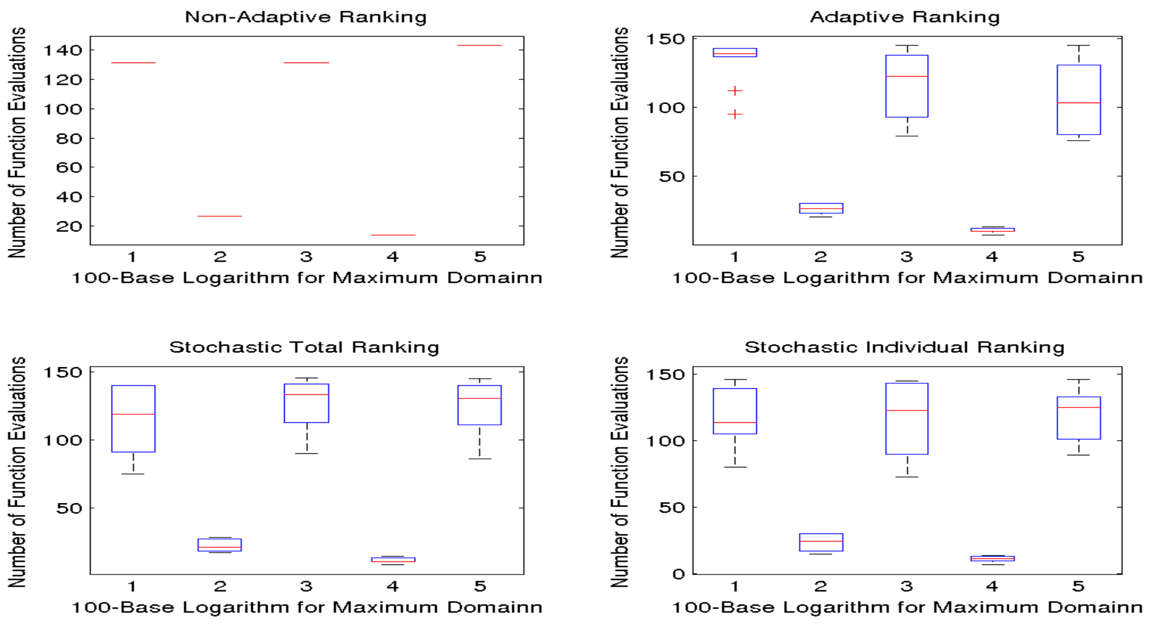

FIGURE 4.10: HCP - Number of Function Evaluations for Michalewicz Function - Second Accuracy Barrier

The graph below shows the execution time for the algorithm to reach the optimal solution within the accuracy error of 1:

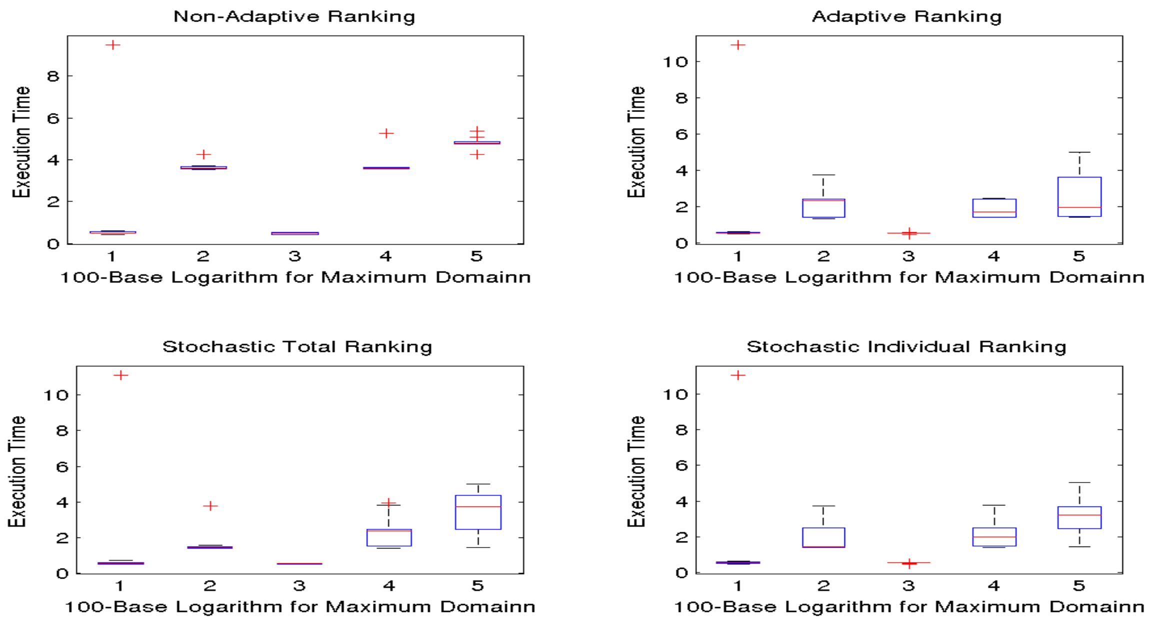

FIGURE 4.11: HCP - Execution Time for Michalewicz Function - Second Accuracy Barrier

4.11.3 Results for Restrigin Function

The last function used for evaluation is Restrigin Function:

$$f(x) = An + \sum_{j=1}^{n}(x_i^2 - A\cos(2\pi x_i)) \tag{4.62}$$

We are testing the function for $A = 10$. The graph below shows the number of function evaluations (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 10:

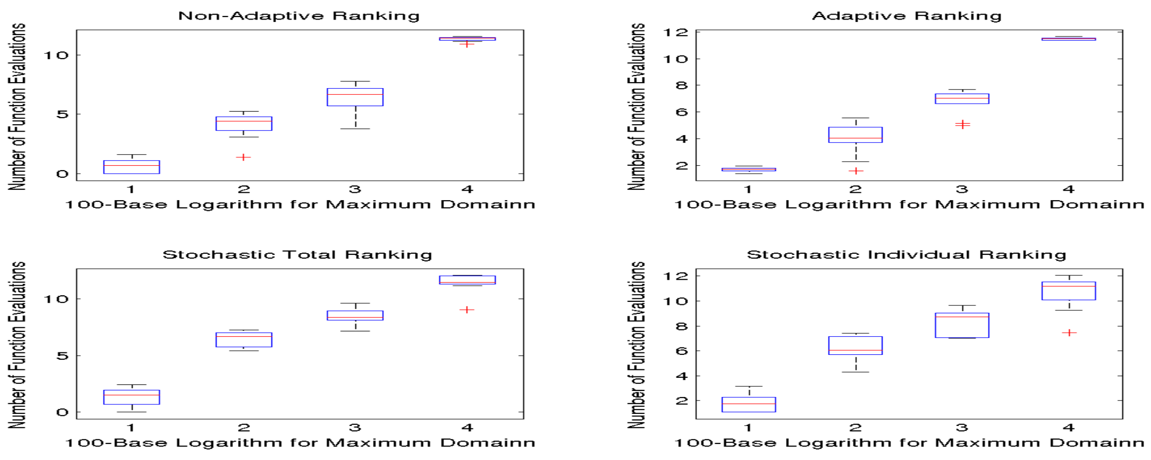

FIGURE 4.12: HCP - Number of Function Evaluations for Restrigin Function - First Accuracy Barrier

The graph below shows the execution time (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 10:

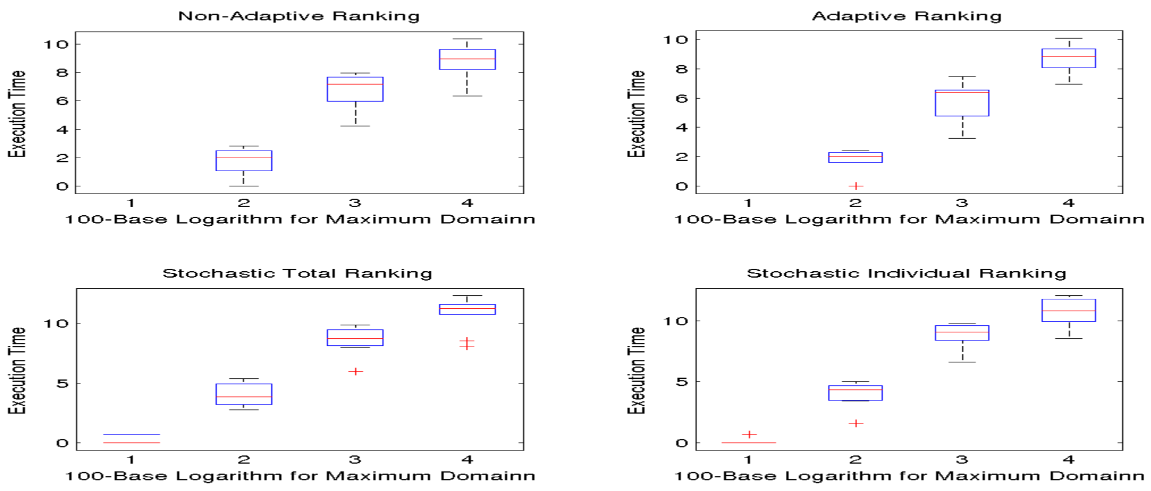

FIGURE 4.13: HCP - Execution Time for Restrigin Function - First Accuracy Barrier

The graph below shows the number of function evaluations (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 1:

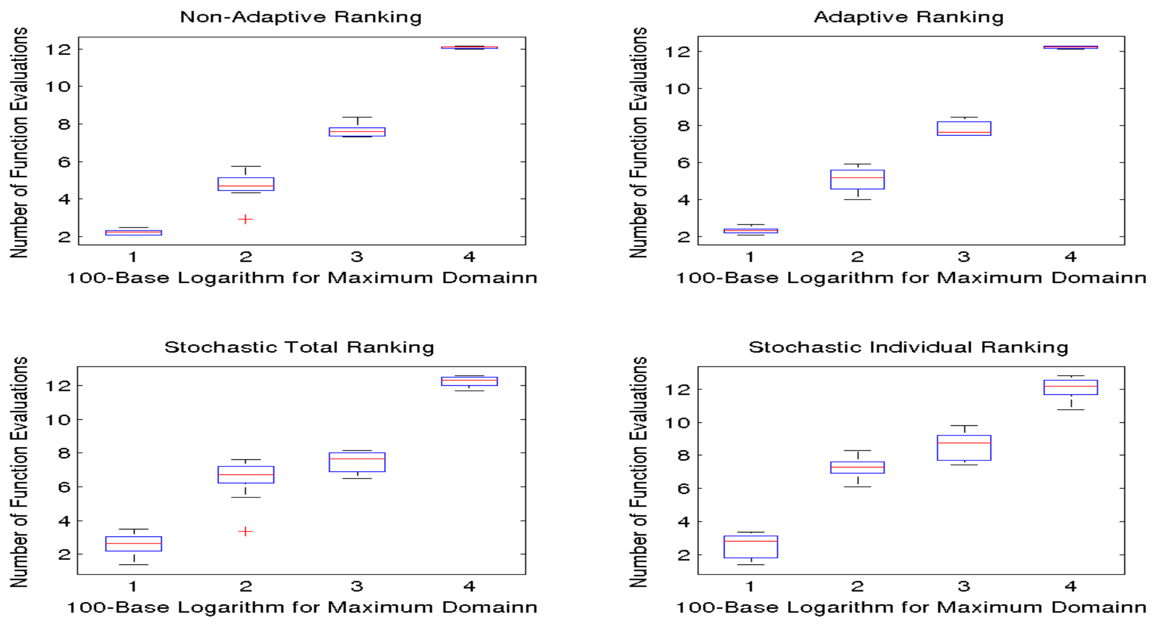

FIGURE 4.14: HCP - Number of Function Evaluations for Restrigin Function - Second Accuracy Barrier

The graph below shows the execution time (in logarithmic scale) for the algorithm to reach the optimal solution within the accuracy error of 1:

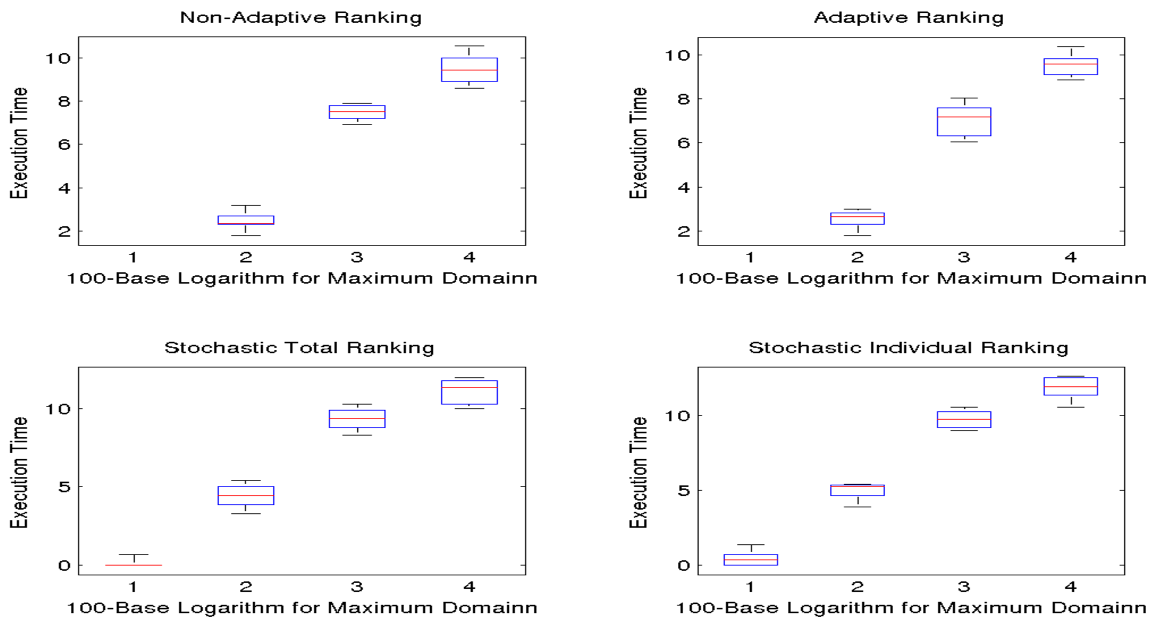

FIGURE 4.15: HCP - Execution Time for Restrigin Function - Second Accuracy Barrier

4.12 Conclusion

4.12.1 Number of function evaluations

The obtained results suggest the superiority of Adaptive and Stochastic ranking methods over Non-Adaptive ranking method. The better performance is most likely caused by the fact that Adaptive and Stochastic ranking methods are not independent of the objective functions, but can be considered greedy, that is it almost always explores the current most promising point. However, the difference in performance is not uniform across different functions and it suggests that the performance of Ranking Methods is likely to depend on the number of local minima in the test functions. However, we need to remember that we tested the methods for $alpha = 0.5$. That variable is used in ranking in the following equation to vary the influence of local information (values of objective function and position in ranking) and the global information (the point with the highest total degree and level):

$$g(x) = (lev(x) + deg(x))^{\alpha} rank(x)^{1-\alpha} \text{ where: } \alpha \in [0,1] \tag{4.63}$$

The fact that we kept the value of α might have influenced the performance of different methods for various functions. Hence, to get a more representative conclusions about the total performance of the methods, we could in the future run more tests, but for different values of α. It would enable us to make more conclusions about the general behavior of the methods independent of the value of α.

4.12.2 Computational Complexity

As is evident from the graphs above, the results for the execution time for different functions and domain sizes follow the patterns seen in the graphs of the number of function evaluations. However, they do not follow them very closely and it also proves that the non-adaptive ranking of hyperbolic cross points is the least computationally exhaustive. The reason for this is the fact that it does not require any meta-data processing for each point such as, for example, keeping sorted list of candidate points with regards to the values of objective function. The other three methods, that is adaptive and stochastic ranking methods require more resources to evaluate the next most appropriate point to evaluate.

The difference stems from the fact that the non-adaptive method ranking does not require much of the computational resources when determining the next point to evaluate. It simply takes another point from the queue of hyperbolic cross points. On the other hand, when other methods try to determine the next point to evaluate, they need to sort all the candidate points. It is not only extremely computationally demanding, but it also puts a lot of strain on memory. Surprisingly, there is not much difference between using the stochastic method and deterministic ranking. However, it might be explained by looking at how probabilities are distributed among different candidate points, for example, in the first method the first point has the probability of being chosen equal to 0.5, while the next one has 0.25, and so on. Hence, it can be expected that in most of the time, half at least, the point is chosen very quickly as, for example, the $10th$ has only the probability of being equal to about: $\frac{1}{2^{10}}$.

4.12.3 Variance of Stochastic Methods

In general, we conclude that the performance of the proposed Stochastic ranking methods is not substantially different from that of Adaptive ranking method. What is more, the means of all three methods are roughly the same and are quite likely different only due to statistical deviations. However, what is evident from the graphs, the Stochastic ranking methods result in greater variance in both execution time as well as the number of function evaluations. What is more, there is no apparent difference between the proposed ways of calculating probabilities for the points in the ranking in Stochastic Ranking method. However, the value and scale of variance are most likely to depend on the method of calculating probabilities for different points in the method. Hence, it could make sense to test other Stochastic method that would enable, for example, uniform sampling of the first k points instead of varying their probabilities based on their position in the original ranking.

4.12.4 Final Remarks

All four tested methods have certain advantages over the others. The the Non-Adaptive method is more appropriate for finding the optimum of the objective functions which do not have highly computationally demanding calculations or have big domain. It is due to the fact that that ranking method does not use any extra ordering and ranking of the candidate points, making the method more computationally scalable. In comparison to this, the Adaptive and Stochastic ranking methods require keeping the ordered list of the candidate points so that their position in the final ranking could be estimated. What is more, although the Adaptive and Stochastic methods seem to have similar performance, the Stochastic Ranking Methods are random and, therefore, they are not deterministic. That property is particularly useful in certain applications, for example, rerunning the same algorithm for the same function might improve the result for Stochastic Ranking method.

However, we note certain weaknesses of the tested methods. In particular, the proposed methods are extremely memory demanding as they need to keep the list of not only all the candidate points, but also of their levels and degrees. In addition to this, our proposed methods might have now bigger parameter space as we need to chose not only the optimal value for the parameter α, but also the optimal value of probability distribution for ranked, candidate points. However, the methods still seem very promising and we think that in the future it could be useful to run more experiments varying the value of α and ways of probability distribution over candidate points to create some heuristics for setting up their values in practical problems.

Chapter 5

Simulated Annealing Function Generalization for Stochastic Gradient-Descent Algorithm

5.1 Stochastic Gradient-Descent Algorithm

In the current version of Stochastic Gradient-Descent algorithm [13], the trajectories are updated in the following way:

$$X(0) = y \in \mathcal{F} \tag{5.1}$$

$$dX(t) = -\nabla f(X(t))dt + \sqrt{2T(t)}dB(t) \tag{5.2}$$

where $B(t)$ is the Brownian motion in $\mathcal{R}^n$ and $\mathcal{F}$ is the feasible set. $T(t)$ is called *annealing schedule* and is used to decrease the effect of random variables over time. $T(t)$ is defined as:

$$T(t) \triangleq \frac{c}{\log(2+t)} \tag{5.3}$$

where c is problem-dependent positive scalar.

5.2 Stochastic Optimization Using the Euler-Maruyana Method

Now, to solve that problem, we need to use numerical experiment to calculate the solution to that Stochastic Differential Equation. We use the Euler-Maruyama method to get for constrained case:

$$X(0) = y \in \mathcal{F} \tag{5.4}$$

$$X(t+1) = \prod \left[X(t) - \nabla f(X(t)) + \mu X(t)^{-1} + \sqrt{2T(t)\Delta t} u \right] \tag{5.5}$$

The projection operator $\prod [x_i] = y_i$ is defined as:

$$\prod [x_i] = \min (u_i, \max (l_i, x_i)) \tag{5.6}$$

where Δt is the length of discrete time step and u is a n-dimensional vector of normally distributed numbers with mean $\mu = 0$ and standard deviation $\sigma = 1$. For unconstrained case, the term P is ignored, that is its values is equal to unity and the iteration is as follows:

$$X(t+1) = X(t) - \nabla f(X(t))\Delta t + \sqrt{2T(t)\Delta t} u \tag{5.7}$$

The algorithm starts by dividing the time length into n steps, that is:

$$\Delta t = \frac{T}{n} \tag{5.8}$$

The next step is to start m different trajectories from each starting point. Each of the trajectories is different due to the randomness of its stochastic term. Every iteration the effect of the stochastic term is decreased. After such a period, we start ranking each of the trajectories. The trajectories are ranked simply by comparing its values, that is $f(x)$ is ranked higher than $f(x^*)$ if and only if $f(x) < f(x^*)$. k of the worst trajectories are then discarded and k of the best trajectories are duplicated. In the next period we just take the produced trajectories and treat them as new starting points. If the vector of highest-ranked trajectories does not change for more than l iterations, we reset noise term. The algorithm terminates when the noise term is too small to make any difference or, in other words, when our stochastic gradient-descent technique turns into linear gradient-descent technique. Another termination condition is when after some number r of noise resets there is no improvement in the best trajectory encountered. All constants are problem-dependent and should be experimented with to receive the best results.

5.3 Convergence of Stochastic Gradient-Descent Algorithm

It has been shown in [14] that for $u : \mathcal{R}^n \to [0, \infty)$, we can choose the annealing schedule function $T(t)$ such that the algorithm converges weakly to a probability measure π around the global minima of U. In such a case, π is the weak limit of the following Gibbs density:

$$p(t,x) = \left[\exp\{-\frac{f(x)}{T(t)}\} \right] \left[\int_{\mathcal{R}^n} \frac{f(x)}{T(t)} dx \right]^{-1} \tag{5.9}$$

In most of the previous papers on Stochastic Gradient-Descent Algorithm, the function used for the annealing schedule has been:

$$T(t) = \frac{c}{\log(t)} \tag{5.10}$$

where c is constant.

Based on the proof given in [14], we can extract the necessary mathematical property that the annealing schedule function should satisfy so that it would still be true that the algorithm weakly converges to the probability π.

5.4 Mathematical Properties of Annealing Schedule Function

In the proof given by [14], it can be noted that the annealing schedule function $\sigma^2(t)$ is used to prove lemma:

$$\int_s^{\beta(s,t)} \frac{\sigma^2(u)}{\sigma^2(s)} du = t \tag{5.11}$$

The function and its properties are used in the last part of the proof, that is:

$$B(t) = \int_0^t |b(Y(s,u))|^2 \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 \frac{1}{\sigma^2(s)} du \tag{5.12}$$

$$\leq \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 du \tag{5.13}$$

$$= \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log u}{\log s} - 1\right)^2 \frac{\log u}{\log s} du \to 0 \tag{5.14}$$

The proof is satisfied by $\sigma^2(t) = \frac{c}{\log t}$ in the following way:

$$\leq \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{u}{s} - 1\right)^2 \frac{\log u}{\log s} du \tag{5.15}$$

$$\leq \frac{constant}{\log(s)} \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.16}$$

$$= \frac{constant}{\log(s)} \frac{(\beta(s,t) - s)^3}{s^2} \tag{5.17}$$

$$\leq \frac{constant}{\log(s)} \to 0 \tag{5.18}$$

5.5 Advantages and disadvantages of standard simulated annealing function

The standard simulated annealing function, $\sigma^2(t) = \frac{c}{\log t}$, has some very useful properties when used for global optimization. First of all, its value decrease relatively slowly and, therefore, it gives the algorithm enough time to discover many regions. In mathematical sense, the variation of the random walk stays relative big and it does not drop so drastically. It suits to most of the problems.

However, as could be noted in the previous section, to prove that the stochastic gradient-descent algorithm converges weakly to π, we do not need to necessarily use the proposed simulated annealing function $\sigma^2(s)$, but could use any other as long as it proves the lemma.

One of the effects of such a change would result in convergence rate, that is we could faster converge to global minimum. Of course, the rate of convergence and, in ours case, simulated annealing function could have various performance for different applications. However, identifying other potential simulated annealing function and testing them could result in improving performance either by increasing the rate of convergence or its error accuracy. Hence, we start by showing some candidates which could be used as a cooling function and, then, we run them on our test functions to see how they affect the performance.

5.6 Mathematical properties of candidates for simulated annealing function

5.6.1 Functions converging to 1

To show the first class of functions of possible candidates for simulated annealing, we start from defining the set of functions $\mathcal{F}_1$, which is a set of all functions monotonically converging to 0:

$$\mathcal{F}_1 : \mathcal{R}^n \to \mathcal{R} : \forall x \in \mathcal{R} : f(x) \geq 1 \wedge f(x) \to 1 \text{ as } x \to \infty \tag{5.19}$$

The possible new candidates for simulated annealing function could be now defined as:

$$\sigma^2(s) = \frac{c}{\log(s)} \frac{1}{f(s)} \tag{5.20}$$

where: $f \in \mathcal{F}$.

Now, to show that such a choice would not change the properties of the stochastic gradient-descent algorithm, the following proof is presented:

$$B(t) = \int_0^t |b(Y(s,u))|^2 \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 \frac{1}{\sigma^2(s)} du \tag{5.21}$$

$$\leq \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 du \tag{5.22}$$

$$= \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log u}{\log s} - 1\right)^2 \frac{\log u}{\log s} du \tag{5.23}$$

$$= \frac{M^2 \log(s)}{c} f(s) \int_0^t \left(\frac{\log u}{\log s} - 1\right)^2 \frac{\log u}{\log s} du \tag{5.24}$$

$$\leq \frac{constant}{\log(s)} f(s) \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.25}$$

From our assumptions, we know that:

$$f(s) \geq 1 \tag{5.26}$$

Hence:

$$\frac{constant}{\log(s)} f(s) \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.27}$$

$$\leq \frac{constant}{\log(s)} f(s) \frac{(\beta(s,t) - s)^3}{s^2} \tag{5.28}$$

$$\leq \frac{constant}{\log(s)} f(s) \tag{5.29}$$

Now, we know because:

$$\lim_{s \to \infty} f(s) = 1 \tag{5.30}$$

and:

$$\lim_{s \to \infty} \frac{constant}{\log(s)} = 0 \tag{5.31}$$

we have:

$$\leq \frac{constant}{\log(s)} f(s) \to 0 \tag{5.32}$$

which completes our proof.

5.6.1.1 Tested example of function converging to 1

In our work, we test one candidate for a simulated annealing function and test its influence. The function, as shown above, needs to satisfy the following assumptions:

- $\forall x \in \mathcal{R} : f(x) \geq 0$
- $\lim_{s\to\infty} f(s) = 1$

The function which satisfies those conditions is:

$$f(x) = \frac{1}{x} + 1 \tag{5.33}$$

and this function is used in our tests. Hence, our simulated annealing function is now:

$$\sigma^2(x) = \frac{c}{\left(\frac{1}{x} + 1\right) \log x} \tag{5.34}$$

5.6.2 Functions of form $\sqrt[n]{log(x)}$

In this subsection, we try to show how we could use the functions of the following form:

$$\sigma^2(x) = \frac{c}{\log(x)} \frac{1}{\sqrt[n]{\log(x)}} \tag{5.35}$$

for all $n \geq 1$.

The propose simulated annealing function would be now:

Now, to show that adding a function of such a from would not change the convergence properties of the stochastic gradient-descent algorithm, the following proof is presented:

$$B(t) = \int_0^t |b(Y(s,u))|^2 \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 \frac{1}{\sigma^2(s)} du \tag{5.36}$$

$$\leq \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log \beta(s,u)}{\log s} - 1\right)^2 du \tag{5.37}$$

$$= \frac{M^2}{\sigma^2(s)} \int_0^t \left(\frac{\log u}{\log s} - 1\right)^2 \frac{\log u}{\log s} du \tag{5.38}$$

$$= \frac{M^2 \log(s)}{c} f(s) \int_0^t \left(\frac{\log u}{\log s} - 1\right)^2 \frac{\log u}{\log s} du \tag{5.39}$$

$$\leq \frac{constant}{\log(s)} f(s) \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.40}$$

From our assumptions, we know that:

$$f(s) = \sqrt[n]{\log(s)} \tag{5.41}$$

Hence:

$$\frac{constant}{\log(s)} \sqrt[n]{\log(s)} \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.42}$$

$$\leq \frac{constant}{\log(s)} \sqrt[n]{\log(s)} \int_0^t \left(\frac{u}{s} - 1\right)^2 du \tag{5.43}$$

$$= \frac{constant}{\log(s)} \sqrt[n]{\log(s)} \frac{(\beta(s,t) - s)^3}{s^2} \tag{5.44}$$

$$\leq \frac{constant}{\log(s)} \sqrt[n]{\log(s)} \tag{5.45}$$

$$\leq \frac{constant}{\sqrt[1-\frac{1}{n}]{\log}(s)} \tag{5.46}$$

Now, we know because:

$$\lim_{s\to\infty} \frac{constant}{\log(s)} = 0 \tag{5.47}$$

then:

$$\lim_{s\to\infty} \frac{constant}{\sqrt[1-\frac{1}{n}]{\log}(s)} = 0 \tag{5.48}$$

we have:

$$\frac{constant}{\sqrt[1-\frac{1}{n}]{\log}(s)} \to 0 \tag{5.49}$$

which completes our proof.

5.6.2.1 Tested example of function converging to 1 of form $\sqrt[n]{\log(x)}$

As has been shown above, the other possible candidate for simulated annealing function can be:

$$\sigma^2(x) = \frac{c}{\sqrt[1+\frac{1}{n}]{\log}(s)} \tag{5.50}$$

where $n \in \mathcal{R}$ and:

$$n \geq 1 \tag{5.51}$$

5.7 Evaluation Methodology

To test the proposed changes to the algorithm, we measure the performance of the algorithm against based on two metrics:

1. Number of function evaluations
2. Error accuracy:
$$\text{error } = f(x_*) - \inf_{x \in [a,b]} f(x') \tag{5.52}$$
where x_* is the obtained solution, and x' is the true global minimum.
3. Execution Time

Each of the method is tested over various domains. We plot whisker plots from the obtained results. The proposed annealing functions are tested on three functions described in the background. The performance of the proposed methods is measured against the original version of Stochastic Gradient Descent. In the presented graphs for each function, the following notation is used:

- Standard Log - results for the standard cooling function: $T(t,c) = \frac{c}{\log(t+2)}$
- Modified Log - results for cooling function: $T(t,c) = \frac{c}{1+\frac{1}{n}\sqrt{\log}(t+2)}$
- Decreasing To One - results for cooling function: $T(t,c) = \frac{c}{\left(\frac{1}{t}+1\right)\log(t+2)}$

5.8 Results

The number of simulations is $n = 100$. We run the algorithm with the given cooling function until it stops. The domain of the algorithm is 2-dimensional, that is $f : \mathcal{R}^2 \to \mathcal{R}$.

5.8.1 Results for Pinter Function

Another test function is Pinter Function $f(x)$ for multi-dimensional vector x, where x^* is the desired global minimizer, is:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1 P_1(x)) + \sin^2(g_2 P_2(x)) \tag{5.53}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{5.54}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{5.55}$$

The values of the variables are:

$$s = 0.05; \tag{5.56}$$

$$g_1 = 20; \tag{5.57}$$

$$g_2 = 10; \tag{5.58}$$

The graph below shows the number of function evaluations (log-scale) before the algorithm stops:

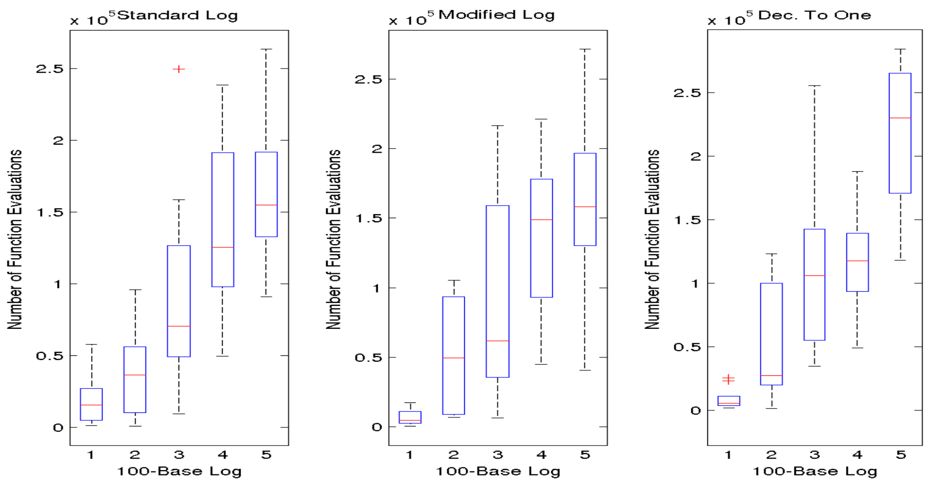

FIGURE 5.1: Number of Function Evaluations for Pinter Function

The graphs below shows the accuracy error (log-scale) achieved by the algorithm and the execution time (log-scale) of the algorithm with different cooling functions:

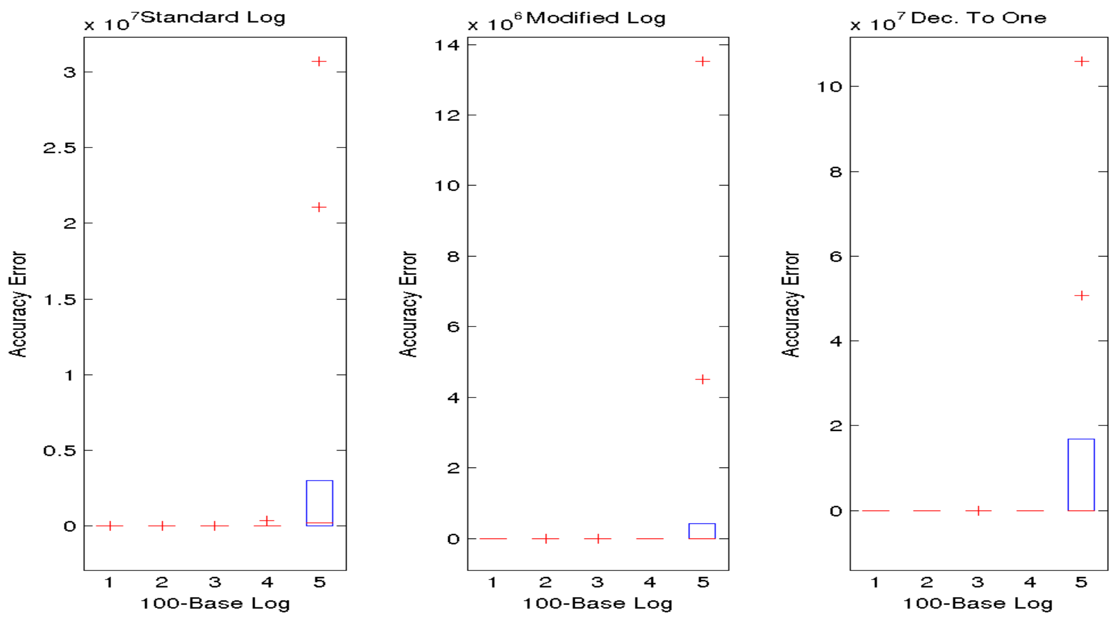

FIGURE 5.2: Accuracy Error for Pinter Function

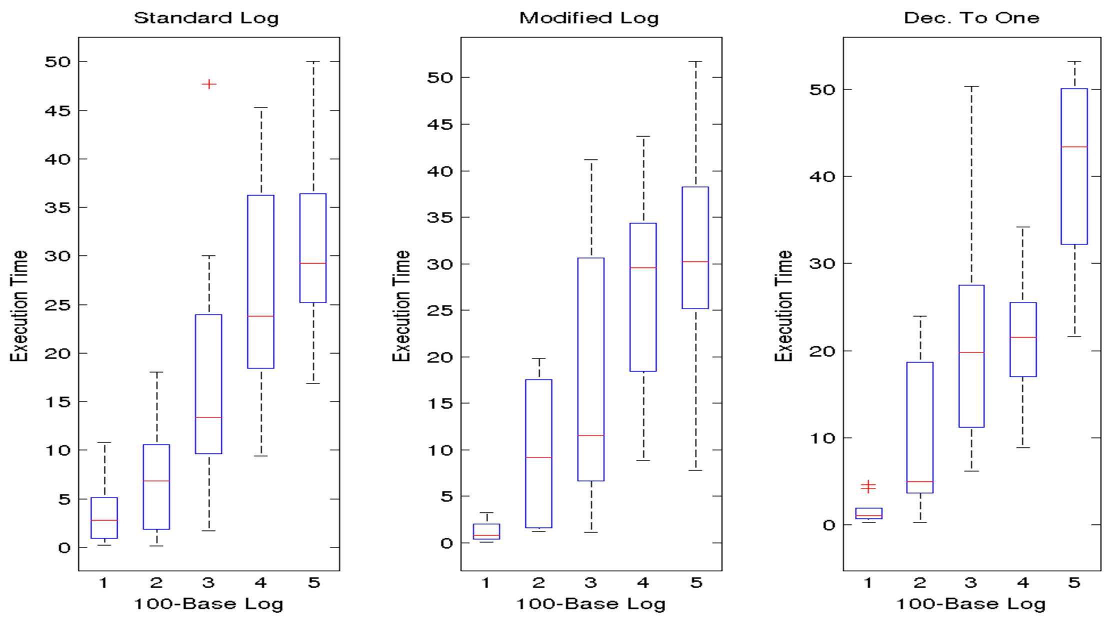

FIGURE 5.3: Execution Time for Pinter Function

5.8.2 Results for Michalewicz Function

The first function we use to evaluate our method is Michalewicz Function. For n-dimensional case, it is of the following form:

$$f(x) = \sum_{j=1}^{n} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{5.59}$$

Now, to use that function in our gradient-descent optimization techniques, we need to calculate its gradient, which is:

$$\frac{\delta f(x)}{\delta x_i} = \cos(x_i \sin^{2m}(\frac{ix_i^2}{\pi}) + 4m\frac{i^2 x_i}{\pi^2} \sin(x_i) \sin^{2m-1}(\frac{ix_i^2}{\pi}) \cos(\frac{ix_i^2}{\pi}) \tag{5.60}$$

The graph below shows the number of function evaluations before the algorithm stops:

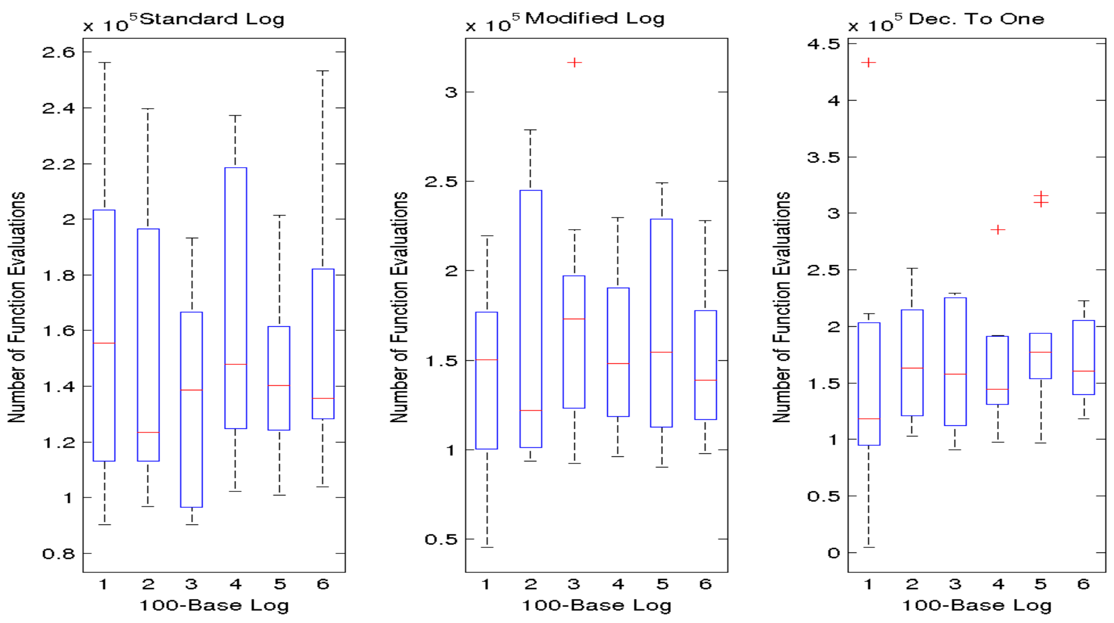

Figure 5.4: Number of Function Evaluations for Michalewicz Function

The graphs below shows the accuracy error achieved by the algorithm and the execution time of the algorithm with different cooling functions:

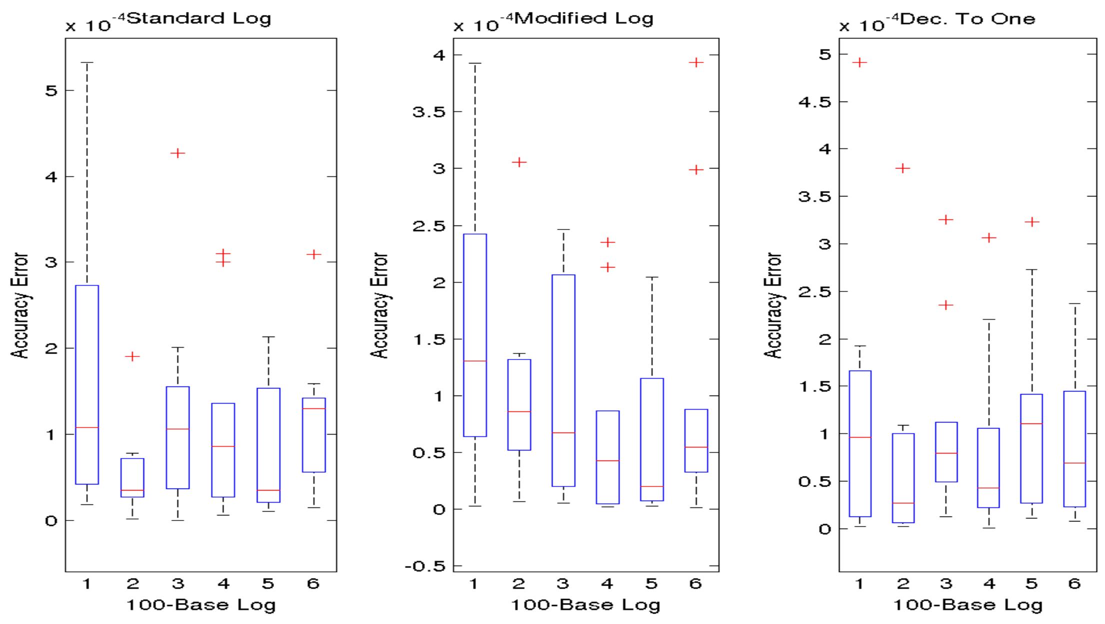

FIGURE 5.5: Accuracy Error for Michalewicz Function

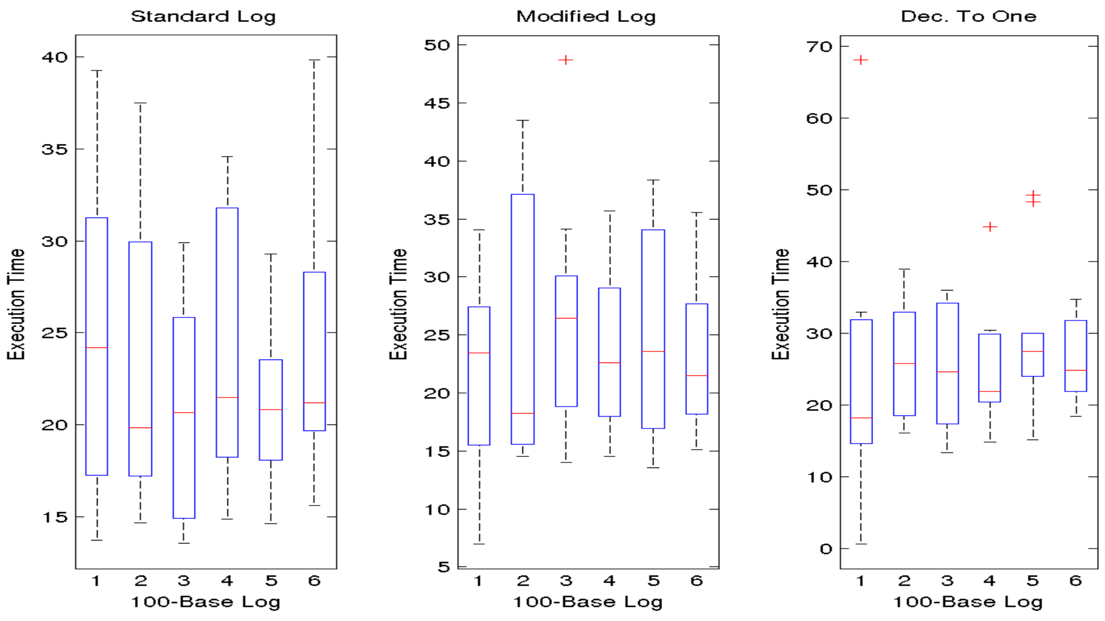

FIGURE 5.6: Execution Time for Michalewicz Function

5.8.3 Results for Restrigin Function

The last function used for evaluation is Restrigin Function:

$$f(x) = An + \sum_{j=1}^{n} (x_i^2 - A\cos(2\pi x_i)) \tag{5.61}$$

Its gradient is:

$$\frac{\delta f(x)}{\delta x_i} = 2x_i + 2A\pi \sin(2\pi x_i)) \tag{5.62}$$

We are testing the function for $A = 10$. The domain is enclosed between 0 and 100^x for each dimension.

The graph below shows the number of function evaluations (log-scale) before the algorithm stops:

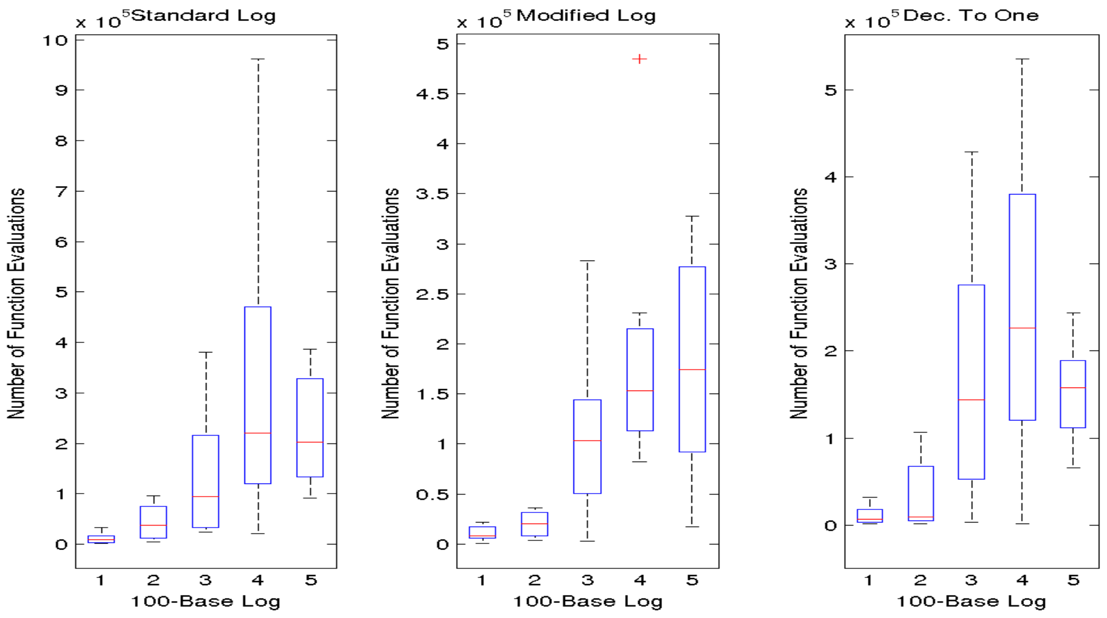

FIGURE 5.7: Number of Function Evaluations for Restrigin Function

The graphs below shows the accuracy error (log-scale) achieved by the algorithm and the execution time (log-scale) of the algorithm with different cooling functions:

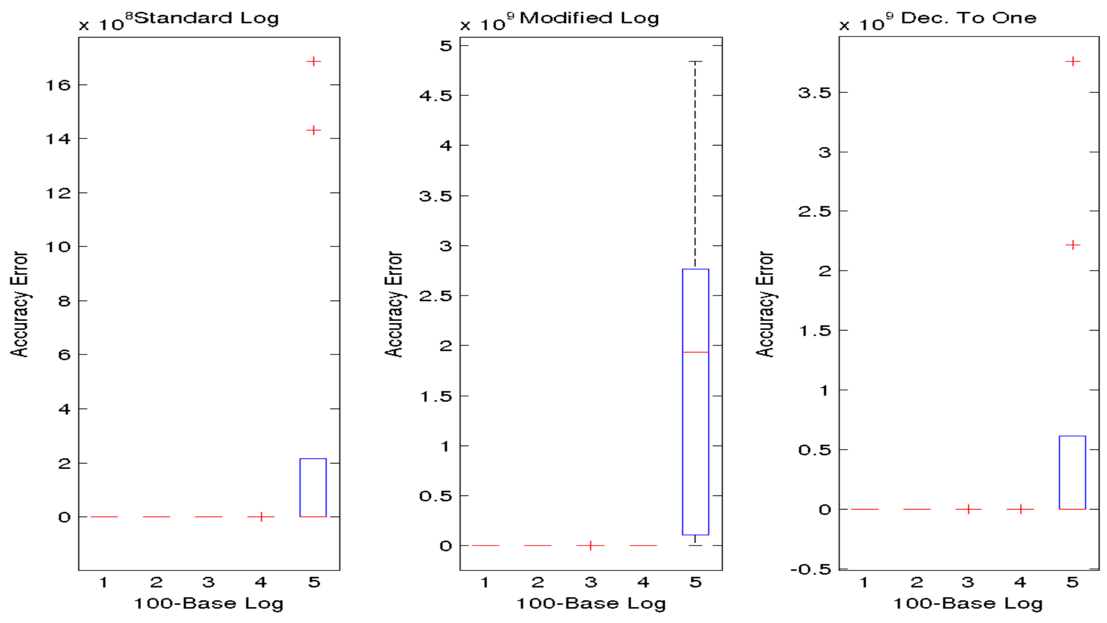

FIGURE 5.8: Accuracy Error for Restrigin Function

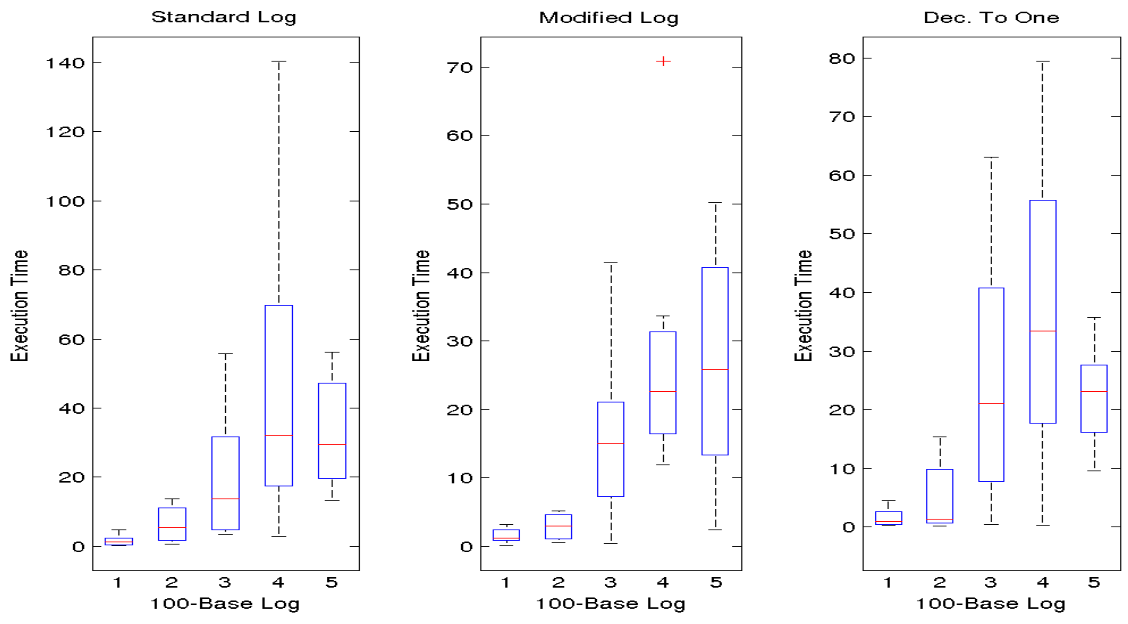

FIGURE 5.9: Execution Time for Restrigin Function

5.9 Conclusions

As can be noted from the results presented above, there is no uniform pattern across various test functions. Hence, we start the analysis of the proposed method by considering each test function separately.

5.9.1 Michalewicz Function

5.9.1.1 Number of Function Evaluations

There is no apparent difference in the number of function evaluations when the standard log cooling function and the modified log cooling function are used. However, when the 'decreasing to one' cooling function is used, it results in the increase in the number of function evaluations. The reason for this could stem from the fact that using that cooling function offers higher decay of the noise factor. Hence, the method would 'cool' more quickly and it could lead more frequently than other methods to local minima. In the case of Michalewicz function, when there are $O(!N)$, where N is the size of dimensions, the number of local minima is very high. Hence, the proposed cooling function might needlessly explore the local minima.

5.9.1.2 Accuracy Error

In this category, the best performing method was the 'decreasing to one' cooling function. The reason for this might be the property discussed above, that is the function tends to explore more local minima and is more likely to find a minimum. However, the difference is not significant and could result as a statistical random difference.

5.9.1.3 Execution Time

As the proposed cooling functions do have similar computational complexity, there is no significant difference in the execution time. The execution times also seem to agree with the pattern of the number of function evaluations, that is the highest execution time is when the 'decreasing to one' cooling function is used.

5.9.1.4 Michalewicz Function Conclusion

Summing up, there is no dominant strategy of choosing one cooling function over the others. As has been discussed above, there is a trade-off between the number of function evaluations or execution time and the accuracy error. When the number of function evaluations is meant to be minimized (for example, when function evaluation is very expensive), then the standard log or the modified log cooling function should be preferred. However, when we are keen on minimizing the accuracy error, we should prefer the 'decreasing to one' cooling function instead.

5.9.2 Pinter Function

5.9.2.1 Number of Function Evaluations

Similar as in the case of Michalewicz Function, the least number of function evaluations resulted from using the standard log or modified log cooling function. However, the difference is not as significant as it was in the case of Michalewicz Function. It might confirm our theory that the different behavior results from the smaller number of local minimum in case of the Pinter Function, that is the cooling functions behave differently when there are different number of local minima.

5.9.2.2 Accuracy Error

All three functions performed extremely well in finding the global minimum and there is no apparent difference between using them.

5.9.2.3 Execution Time

The pattern is the same as for the number of Function Evaluations.

5.9.2.4 Pinter Function Conclusion

The results for the Pinter Function seem to confirm our conclusions for Michalewicz Function, that is the number of local minima influences which cooling function should be used. However, as there is no apparent difference in accuracy error between the proposed cooling methods, and the standard log and modified log cooling functions perform better than the 'decreasing to one' cooling function in terms of the number of function evaluations and execution time, we conclude that for the Pinter Function it is not preferred to used the 'decreasing to one' cooling function.

5.9.3 Restrigin Function

5.9.3.1 Number of Function Evaluations

Similar as in the case of both Michalewicz Function and Pinter Function, the least number of function evaluations is found when using the standard log or modified log cooling function.

5.9.3.2 Accuracy Error and Execution Time

The substantial better accuracy error and smaller number of function evaluations are achieved when using the 'decreasing to one' cooling function.

5.9.3.3 Restrigin Function Conclusion

The results for the Restrigin Function seem to confirm our conclusions from the previous functions: the standard log and modified log cooling functions lower the total number of function evaluations, but the 'decreasing to one' cooling function results in lower accuracy error.

5.9.4 Final Conclusion

The final conclusion might suggest that there is a difference in performance when using different cooling functions and each of the proposed functions offer some trade-off when using for specific function. In particular, we could use the standard log and modified log cooling functions interchangeably without any significant impact on performance in case of our test functions. In general, those two cooling functions performed better than the 'decreasing to one' cooling function. However, in the situation when the number of local minima is substantial and we are keen on sacrificing the execution time or number of function evaluations for the sake of accuracy, we might consider using the 'decreasing to one' cooling function.

The results from testing the proposed cooling functions show that there is a variety of other cooling function, except for the most widely used: $T(t,c) = \frac{c}{log(t+2)}$ that retains its mathematical rigorousness, but offering slightly different behavior. It seems that the specific behavior of different cooling functions might depend on the specific problem. Hence, using different cooling functions might be used as a tuning of algorithm to the specific problem.That result shows the potential for method and the usefulness of analyzing various approaches to cooling function.

However, one of the biggest weaknesses of the proposed method is that we have only heuristics how they might work for different problem. Hence, it might be necessary to run different methods for new problems to asses their performance. In addition to this, another weakness is that no cooling function might be considered a dominating one in the sense that it performs better than another considering all statistics. Therefore, we might reason if, for example, we are keen on sacrificing the execution time for the sake of achieving the better result.

We also think in the future works it might be useful to run the proposed methods on more test problems to determine their exact properties and behavior under various circumstances in order to create heuristics or more mathematically rigid methods for choosing the most appropriate annealing function.

Chapter 6

Stochastic Gradient Descent with Point Repetition Avoidance

6.1 Problem Introduction

In the previous chapter, we consider Stochastic Gradient Descent Algorithm [13]. It is as follows:

$$X(0) = y \in \mathcal{F} \tag{6.1}$$

$$dX(t) = -\nabla f(X(t))dt + \sqrt{2T(t)}dB(t) \tag{6.2}$$

where $B(t)$ is the Brownian motion in $\mathcal{R}^n$. $T(t)$ is called *annealing schedule* and is used to decrease the effect of random variables over time. $T(t)$ is defined as:

$$T(t) \triangleq \frac{c}{log(2+t)} \tag{6.3}$$

where c is problem-dependent positive scalar.

However, one of the problems of the described algorithm is that currently there is no mechanism to prevent the algorithm from exploring the already explored areas. The next point at time t has the following probability distribution:

$$x_{t+1} \sim \mathcal{N}\left(x_t - \nabla f(x_t), 2T(t)\right) \tag{6.4}$$

In has been proven in [14] that the method is weakly convergent. As can be noted, every point has non-zero probability of being at any other place, even coming back to the previous position.

6.2 Avoiding Point Repetition

Avoiding exploring the already explored regions should not affect the problem accuracy, but it might decrease the number of function evaluations. The proposed method of avoiding the repeated exploration is adding the following term to the stochastic differential equation:

$$K(t) = \alpha \int_{\max(0,t-C)}^{t} (x(t) - x(s)) \exp\left(\frac{\|x(t) - x(s)\|_2^2}{\beta}\right) ds \tag{6.5}$$

where the term C is the limit of number of points used in the algorithm. One of the main advantages of that is that we do not need to keep track of all the points from the start, but we could prune them and keep only C latest ones of them. In addition to this, it is supposed to prevent the algorithm from getting stuck and creating local minima towards which the trajectory is drawn.

The stochastic differential equation evaluated at each time is now:

$$dX(t) = -\nabla f(X(t))dt + K(t) + \sqrt{2T(t)}dB(t) \tag{6.6}$$

$$dX(t) = -\nabla f(X(t))dt + \alpha \int_{\max(0,t-C)}^{t} (x(t) - x(s)) \exp\left(\frac{\|x(t) - x(s)\|_2^2}{\beta} ds\right) + \sqrt{2T(t)}dB(t) \tag{6.7}$$

The next point has now the following distribution:

$$x_{t+1} \sim \mathcal{N}\left(x_t - \nabla f(x_t) + K(t), 2T(t)\right) \tag{6.8}$$

$$x_{t+1} \sim \mathcal{N}\left(x_t - \nabla f(x_t) + \alpha \int_{max(0,t-C)}^{t} (x(t) - x(s)) \exp\left(\frac{\|x(t) - x(s)\|_2^2}{\beta}\right) ds, 2T(t)\right) \tag{6.9}$$

6.2.1 Weighted Trajectory Points

Another tested version of the proposed modification takes into account how current the given point is, that is it weights the influence of the points trajectory encountered:

$$K(t) = \alpha \int_{max(0,t-C)}^{t} \frac{t-s}{C} (x(t) - x(s)) \exp\left(\frac{\|x(t) - x(s)\|_2^2}{\beta}\right) ds \tag{6.10}$$

Hence, the next point in the trajectory is now defined as:

$$x_{t+1} \sim \mathcal{N}\left(x_t - \nabla f(x_t) + \alpha \int_{max(0,t-C)}^{t} \frac{t-s}{C}(x(t)-x(s)) \exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) ds, 2T(t)\right) \tag{6.11}$$

6.3 Mathematical properties of $K(t)$

Intuitively, the additional term $K(t)$ can be considered to be a dynamic bias against the explored regions, for example, if the algorithm is between two relatively explored areas, the term $K(t)$ will be close to zero and the trajectory is not modified. Similarly, when a point is well-explored areas, the term $K(t)$ also will be zero and the trajectory is also evaluated as it was in the original version of the algorithm. However, once some point is reached while the other point have already been explored, the trajectory is discouraged by continuing the exploration of that region.

More rigorously, the algorithm stops when the convergence is reached, that is we have:

$$|x_{t+1} - x_t| < \epsilon \tag{6.12}$$

It means that the value of $K(t)$ should decrease to zero for the algorithm to stop:

$$\lim_{t \leftarrow \infty} K(t) = 0 \tag{6.13}$$

However, the value of $K(t)$ for some point $x(t)$ whenever the trajectory has evenly explored the points around the point. Therefore, the trajectory is encouraged to explore the region more evenly, which means avoiding exploring the already explored regions.

6.4 Numerical Integration

To evaluation the term $K(t)$, we need to calculate the following integral for the first version of the proposed change:

$$K(t) = \alpha \int_{\max(0,t-C)}^{t} (x(t)-x(s)) \exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) ds \tag{6.14}$$

For the weighted version of the proposed change, we have:

$$K(t) = \alpha \int_{\max(0,t-C)}^{t} \frac{t-s}{C}(x(t)-x(s)) \exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) ds \tag{6.15}$$

However, the integrated function is not continuous, hence we need to use a numerical algorithm to evaluate it. In this dissertation, we use the following method for integral evaluation:

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2}+\sum_{k=1}^{n-1}\left(f\left(a+k\frac{b-a}{n}\right)\right)+\frac{f(b)}{2}\right) \tag{6.16}$$

In the given algorithm, the size of time-step k is 1. Furthermore, at time t we have only t number of points. Hence, we have $n = t$ and the integral is evaluated as:

$$K(t) = \alpha\int_0^t (x(t)-x(s))\exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) \tag{6.17}$$

and:

$$k(s) = (x(t)-x(s))\exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) \tag{6.18}$$

or:

$$k(s) = \frac{t-s}{C}(x(t)-x(s))\exp\left(\frac{\|x(t)-x(s)\|_2^2}{\beta}\right) \tag{6.19}$$

$$\frac{b-a}{n}\left(\frac{f(a)}{2}+\sum_{k=1}^{n-1}\left(f\left(a+k\frac{b-a}{n}\right)\right)+\frac{f(b)}{2}\right) \approx \left(\frac{f(0)}{2}+\sum_{k=1}^{n-1}\left(f\left(k\right)\right)+\frac{f(t)}{2}\right) \tag{6.20}$$

$$K(t) \approx \frac{k(0)}{2}+\sum_{s=1}^{n-1}k\left(s\right)+\frac{k(t)}{2} \tag{6.21}$$

6.5 Euler Numerical Solution to SDE

Using the Euler solution to Stochastic Differential Equation, we have:

$$x_{k+1} = x_k - \nabla f(X(t))dt + K(t) + \sqrt{2T(t)}dB(t) \tag{6.22}$$

$$x_{k+1} = x_k - \nabla f(X(t))dt + \alpha\int_{\max(0,t-C)}^t k(s)ds + \sqrt{2T(t)}dB(t) \tag{6.23}$$

Substituting the numerical solution to the integral, the trajectory is updated at every iteration with the following equation:

$$x_{k+1} = x_k - \nabla f(X(t)) + \left(\frac{k(0)}{2}+\sum_{s=1}^{n-1}k\left(s\right)+\frac{k(t)}{2}\right) + \sqrt{2T(t)}dB(t) \tag{6.24}$$

6.6 Evaluation Methodology

In this chapter, we run the experiments to determine whether or not the addition of the proposed term influence the number of function evaluations and the trajectory taken by the algorithm. We run simulations on the three test functions and take note of the following outcomes:

1. Number of function evaluations
2. Execution time
3. Accuracy Error
4. Number of repeated points within some interval, that is the cardinality of the following set: $| \{x_i s.t. \exists x_j, i \neq j, |x_i - x_j| < \epsilon\} |$

The results are compared against the original Stochastic Gradient Descent Algorithm. We run two version of the proposed added term:

1. Time-weighted points
2. Equally weighted points

From the initial tests, we have determined that the most promising values of α and β are:

- $\alpha = -1$
- $\beta = -2$

Hence, we run simulations using those values. We run $N = 100$ for each function to get an idea about the performance of the proposed changes. The epsilon value is $\epsilon = 0.1$.

In graphs produced from results, we are going to use the following notation for marking what method was used to produce given results:

1. **Std. SDE**: Standard Stochastic Gradient Descent algorithm. It is the original version of the algorithm as presented in [**?**].
2. **Equal K(t)**: the algorithm with added the integral term $K(t)$, where each of the past points contributes equally to the final value of $K(t)$.
3. **Weight K(t)**: the algorithm with added the integral term $K(t)$, where each of the past points contributes the final value of $K(t)$ proportionally to how recent it is.

6.7 Results

The number of iterations is $n = 100$. Every iteration we run the algorithm with the given cooling function until it stops. The domain of the algorithm is 2-dimensional, that is $f : \mathcal{R}^2 \to \mathcal{R}$.

6.7.1 Pinter Function

Another test function is Pinter Function $f(x)$ for multi-dimensional vector x, where x^* is the desired global minimizer, is:

$$f(x) = s\sum_{i=1}^{n}(x_i - x_i^*)^2 + \sin^2(g_1P_1(x)) + \sin^2(g_2P_2(x)) \tag{6.25}$$

$$P_1(x) = \sum_{i=1}^{n}(x_i - x_i^*)^2 + \sum_{i=1}^{n}(x_i - x_i^*) \tag{6.26}$$

$$P_2(x) = \sum_{i=1}^{n}(x_i - x_i^*) \tag{6.27}$$

The values of the variables are:

$$s = 0.05; \tag{6.28}$$

$$g_1 = 20; \tag{6.29}$$

$$g_2 = 10; \tag{6.30}$$

The graph below shows the number of function evaluations the algorithm takes:

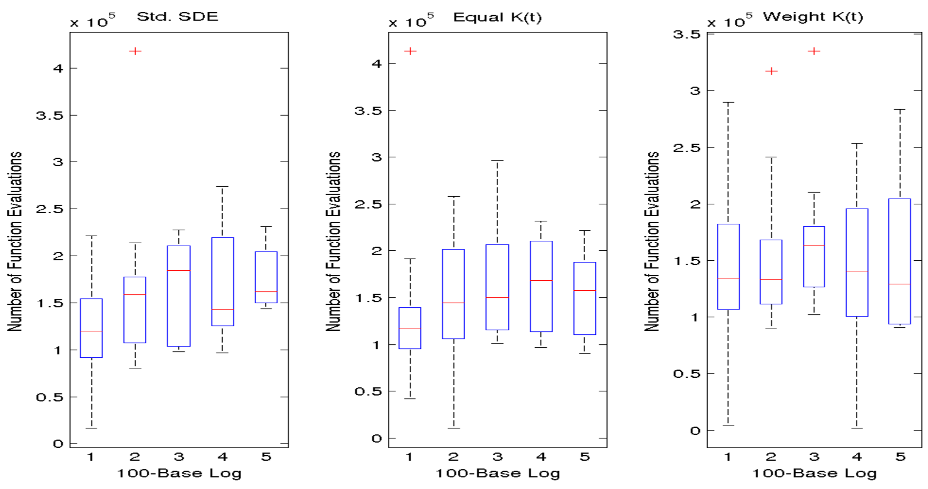

FIGURE 6.1: Number of Function Evaluations for Pinter Function

The graphs below shows the accuracy error achieved by the algorithm and the execution time of the algorithm with different cooling functions:

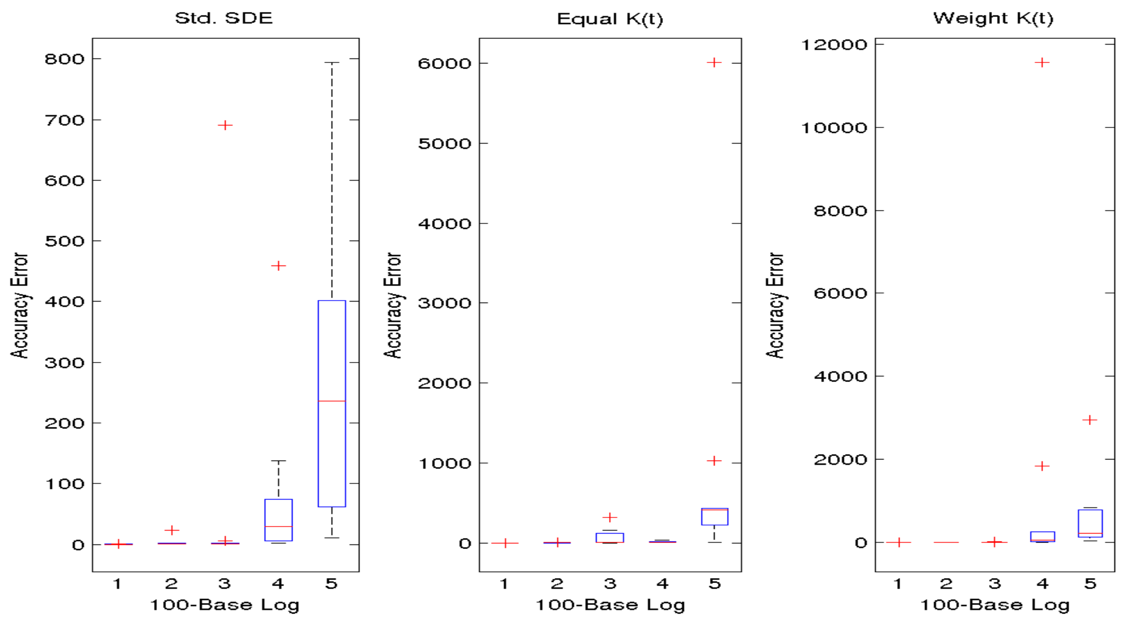

FIGURE 6.2: Accuracy Error for Pinter Function

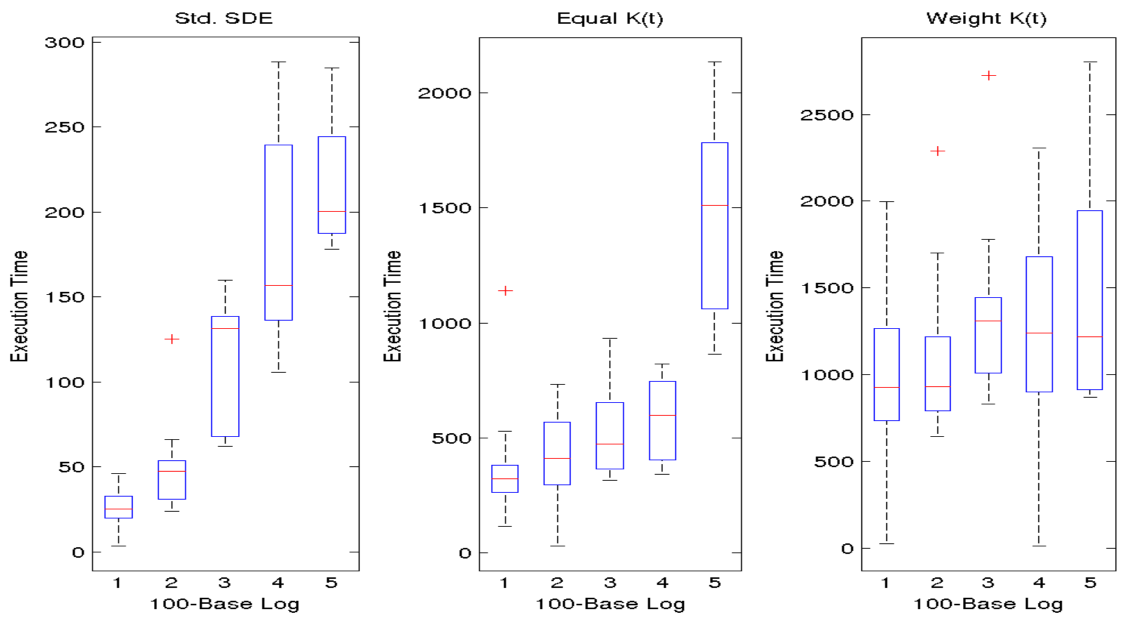

FIGURE 6.3: Execution Time for Pinter Function

The last graph presents the number of unique points explored by the algorithm:

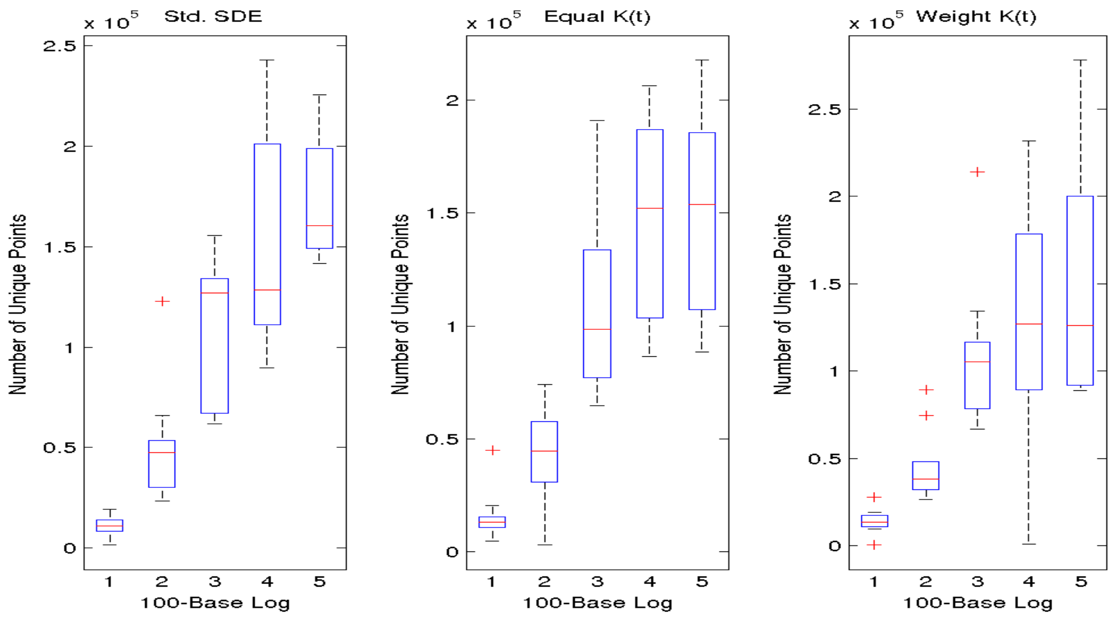

FIGURE 6.4: Number of unique points for Pinter Function

6.7.2 Results for Michalewicz Function

The first function we use to evaluate our method is Michalewicz Function. For n-dimensional case, it is of the following form:

$$f(x) = \sum_{j=1}^{n} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{6.31}$$

Now, to use that function in our gradient-descent optimization techniques, we need to calculate its gradient, which is:

$$\frac{\delta f(x)}{\delta x_i} = \cos(x_i \sin^{2m}(\frac{ix_i^2}{\pi}) + 4m\frac{i^2 x_i}{\pi^2} \sin(x_i) \sin^{2m-1}(\frac{ix_i^2}{\pi}) \cos(\frac{ix_i^2}{\pi}) \tag{6.32}$$

The graph below shows the number of function evaluations the algorithm takes:

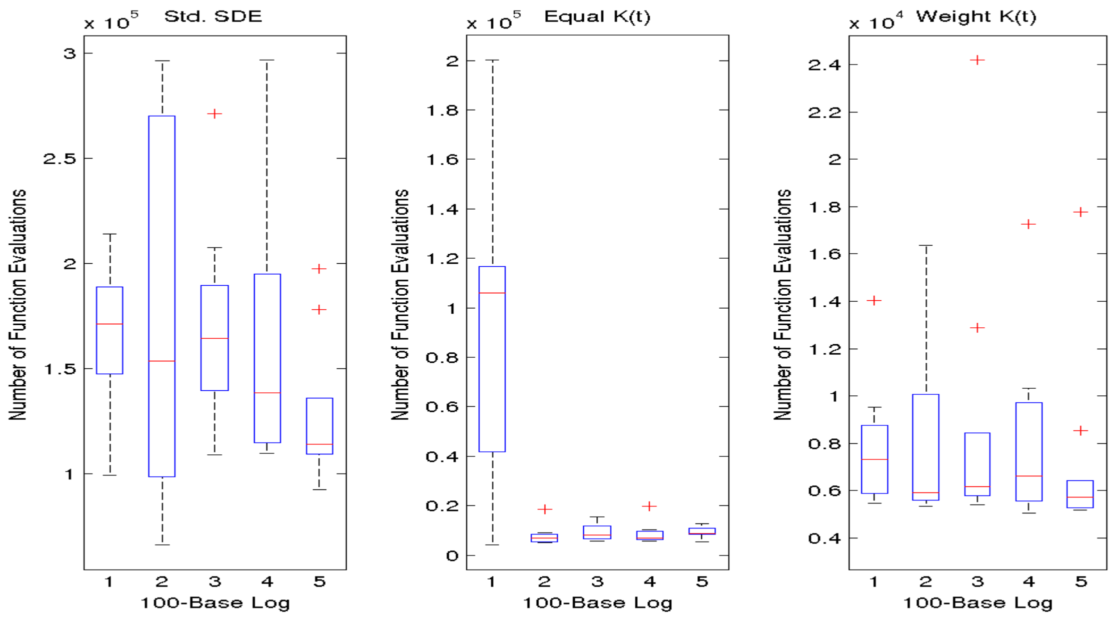

FIGURE 6.5: Number of Function Evaluations for Michalewicz Function

The graphs below shows the accuracy error achieved by the algorithm and the execution time of the algorithm with different cooling functions:

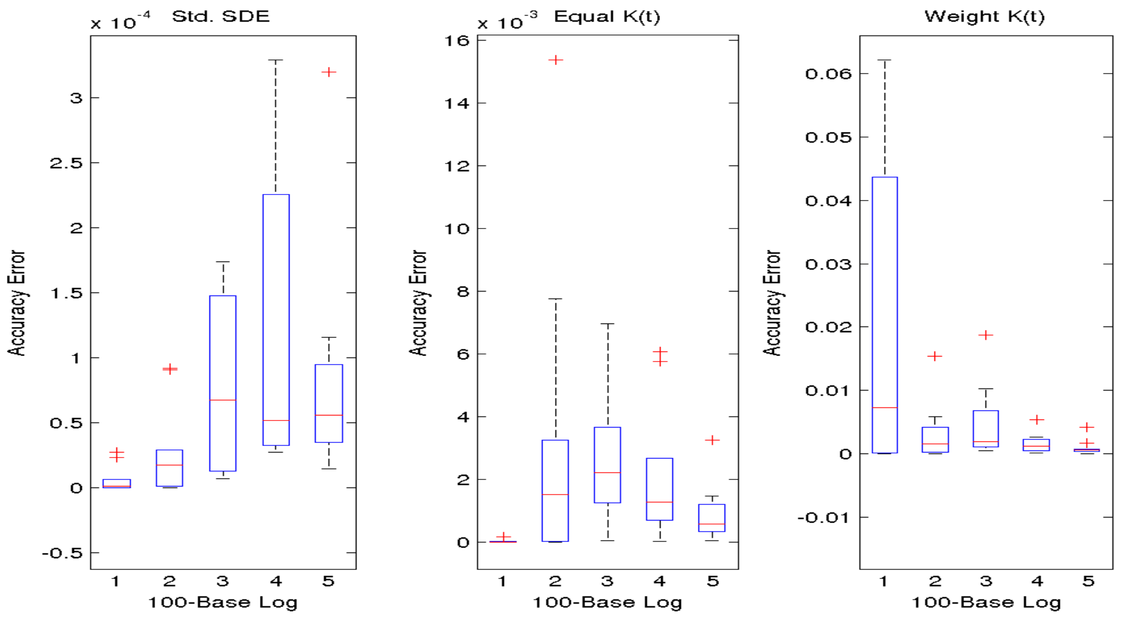

FIGURE 6.6: Accuracy Error for Michalewicz Function

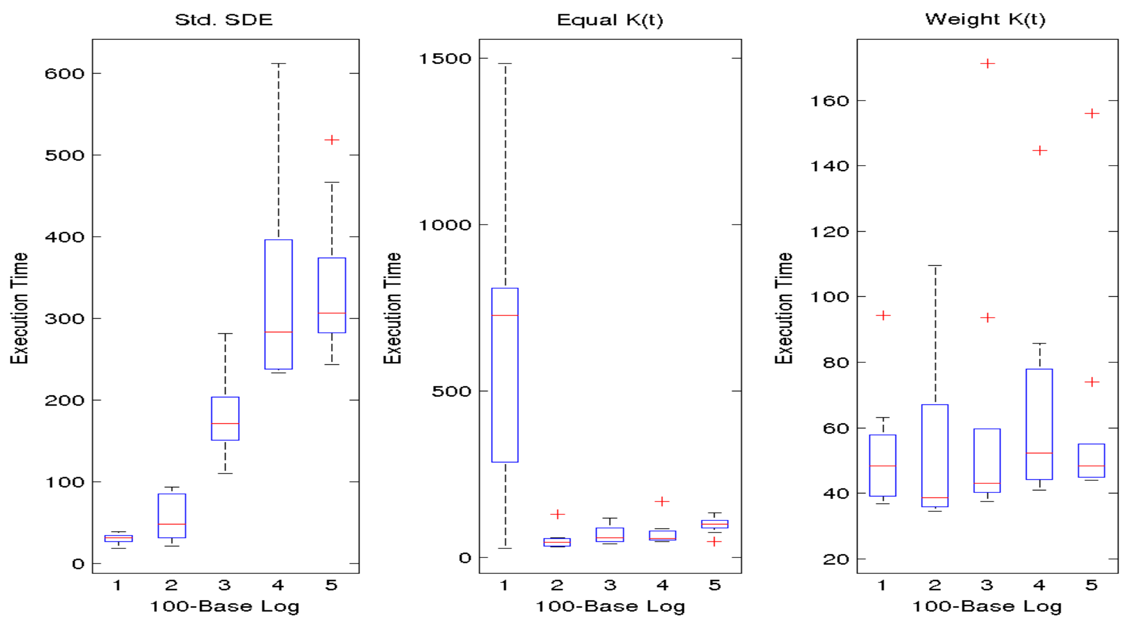

FIGURE 6.7: Execution Time for Michalewicz Function

The last graph presents the number of unique points explored by the algorithm:

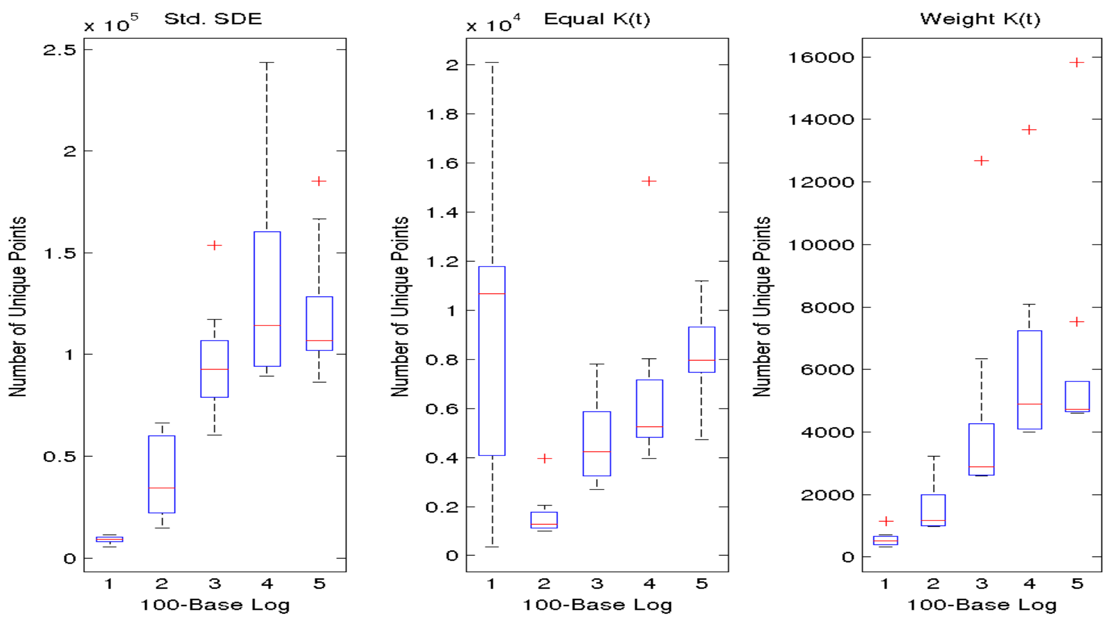

FIGURE 6.8: Number of unique points for Michalewicz Function

6.7.3 Restrigin Function

The last function used for evaluation is Restrigin Function:

$$f(x) = An + \sum_{j=1}^{n}(x_i^2 - A\cos(2\pi x_i)) \tag{6.33}$$

Its gradient is:

$$\frac{\delta f(x)}{\delta x_i} = 2x_i + 2A\pi \sin(2\pi x_i)) \tag{6.34}$$

We are testing the function for $A = 10$. The domain is enclosed between 0 and 100^x for each dimension.

The graph below shows the number of function evaluations the algorithm takes:

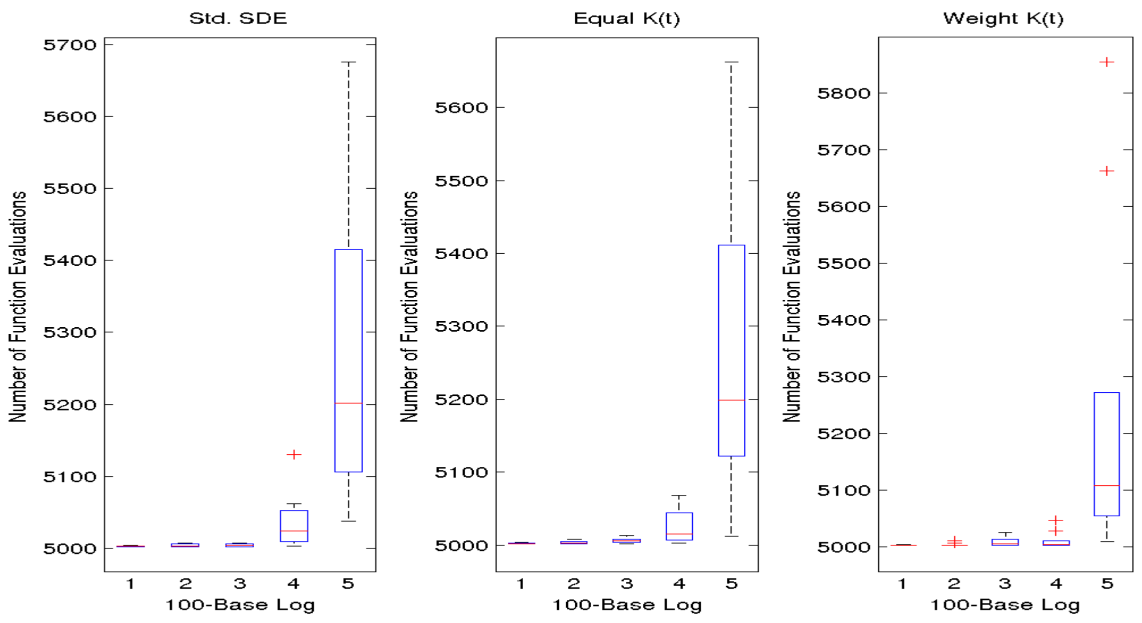

FIGURE 6.9: Number of Function Evaluations for Restrigin Function

The graphs below shows the accuracy error achieved by the algorithm and the execution time of the algorithm with different cooling functions:

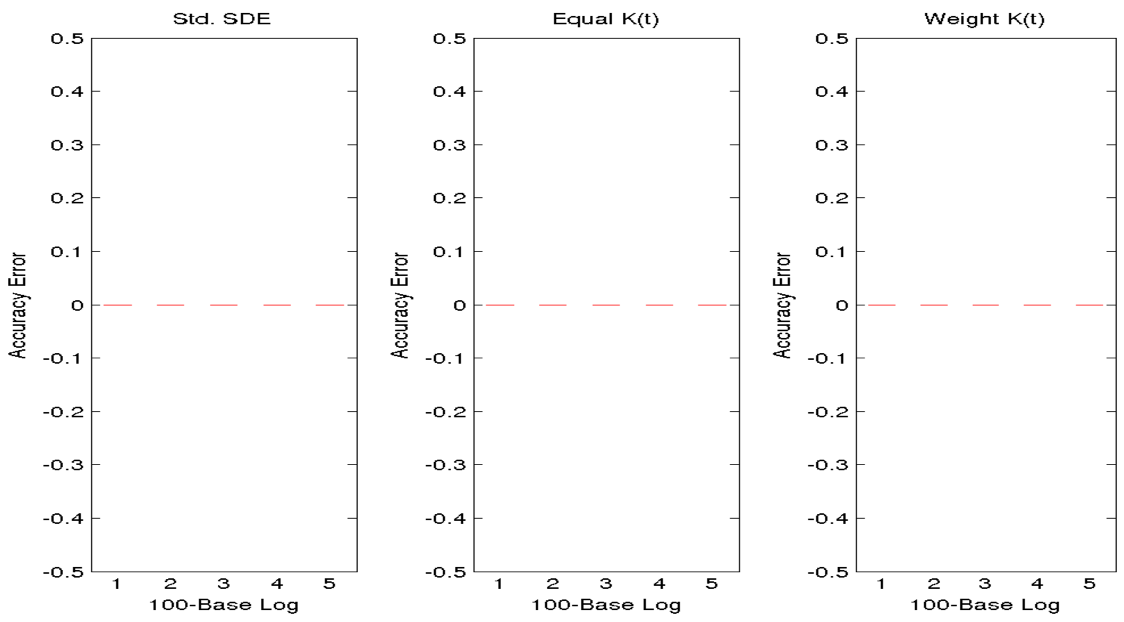

FIGURE 6.10: Accuracy Error for Restrigin Function

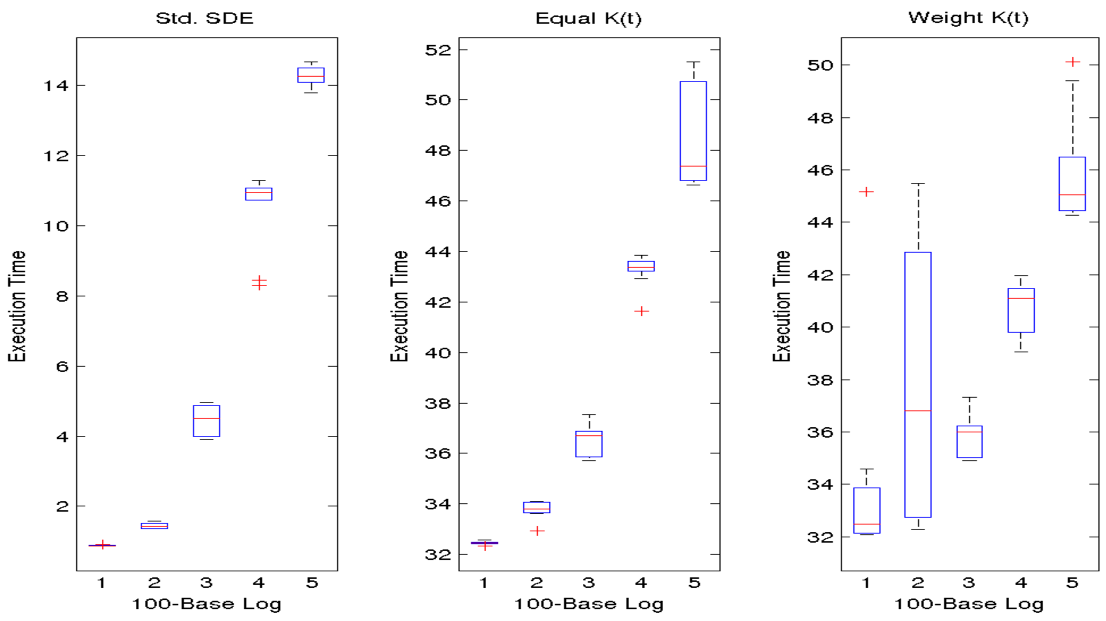

FIGURE 6.11: Execution Time for Restrigin Function

The last graph presents the number of unique points explored by the algorithm:

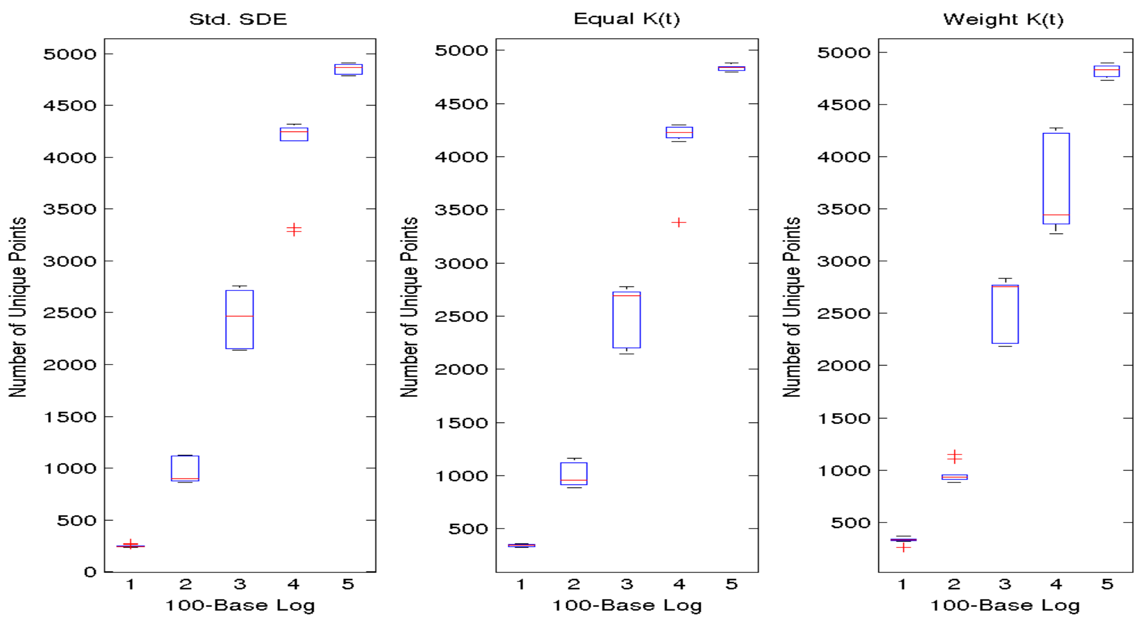

FIGURE 6.12: Number of unique points for Restrigin Function

6.8 Conclusions

6.8.1 Observations

For each of the tested functions, the number of function evaluations for different sizes of domain is approximately the same. There is no general advantage of using one method over another. However, it can be noted that every method performs differently for different domain. In particular, adding the point avoidance term $K(t)$ performs better when using bigger domains in terms of error accuracy, that is it is more likely to find the better solution before the termination conditions are met. Another trend we spotted is the fact that the bigger the size of domain, the better the proposed methods perform. However, as can be noted, that improvement comes at some price: execution time increase almost ten times. That result is not surprising as we need to iterate through $C = 1000$ last points to evaluate the integral term $K(t)$. It is both CPU- and memory-intensive. Those conclusions are based on the results from Pinter and Michalewicz functions. In the case of Restrigin function, the algorithms almost always find the true, global minimum for the given termination conditions.

Another trend that could be noticed is that there is not much improvement in the unique number of points evaluated when we add the proposed integral term $K(t)$. Considering that the accuracy error has improved, that result is surprising. However, it could be explained by the fact that although the number of unique points stays roughly the same, the distribution of the evaluated regions changes, that is the trajectory is encouraged by the term $K(t)$ to evaluate points that are considerably separate from each other while the original version might result in evaluating many different points, but in the same neighborhood or convexity region. This result shows that the trajectory in the original version of the algorithm is more suited for exploring local convexity areas, while the proposed term $K(t)$ encourages points that are more evenly distributed across domains.

6.8.2 Locality Problem of Accumulated Historic Points for $K(t)$

In the resultant graphs, it is visible that although the means of accuracy error in case of the proposed changes are better than those of the original algorithm, the proposed changes have much bigger standard deviation. Although on average the proposed methods outperform the original algorithm, there are also cases that when their performance is much worse. To find out the reason for this, we ran another set of simulations. We focus on Pinter function, and notice that the proposed method might get stuck. The following graph shows a couple of initial iterations of the first proposed version of the algorithm when its trajectory gets stuck. The results are taken for domain: $0 - 10$ in one dimensional problem. The domain is deliberately small to see how it affects the trajectory. The red dot is the current position of the trajectory at a given iteration. The blue line represents the Pinter function.

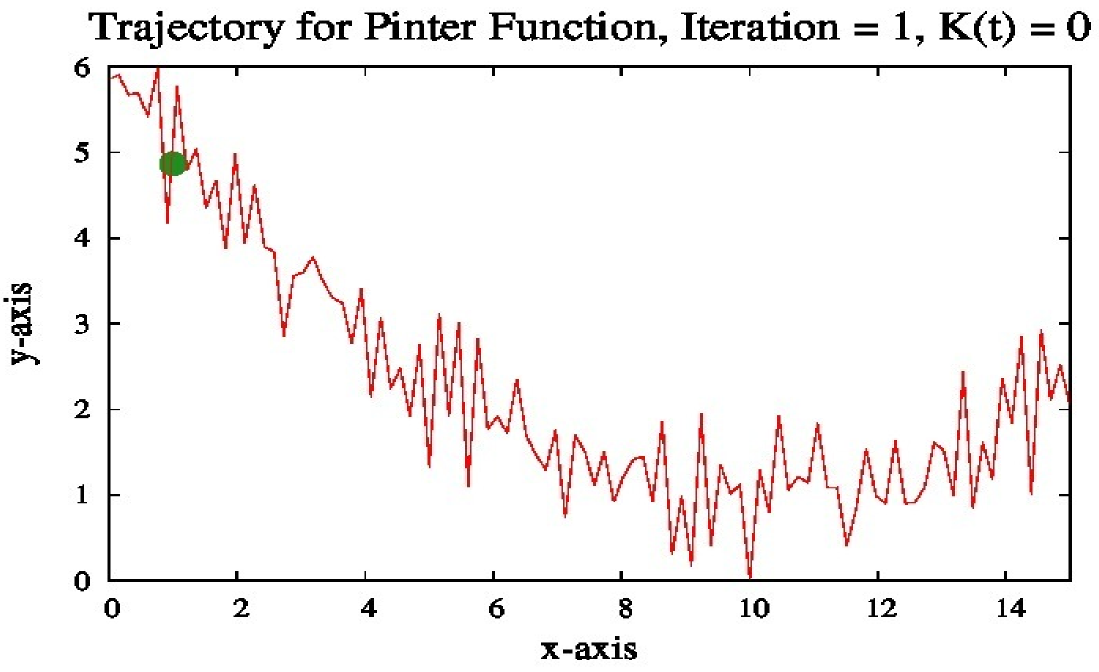

Figure 6.13: Algorithm Trajectory - Step 1

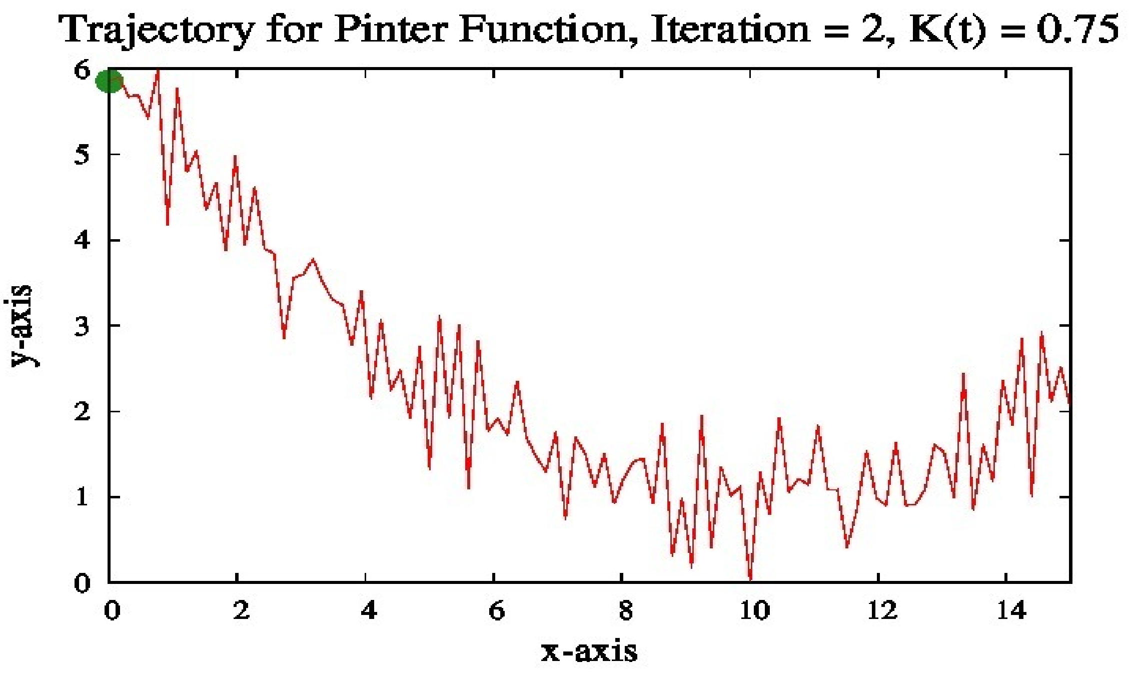

FIGURE 6.14: Algorithm Trajectory - Step 2

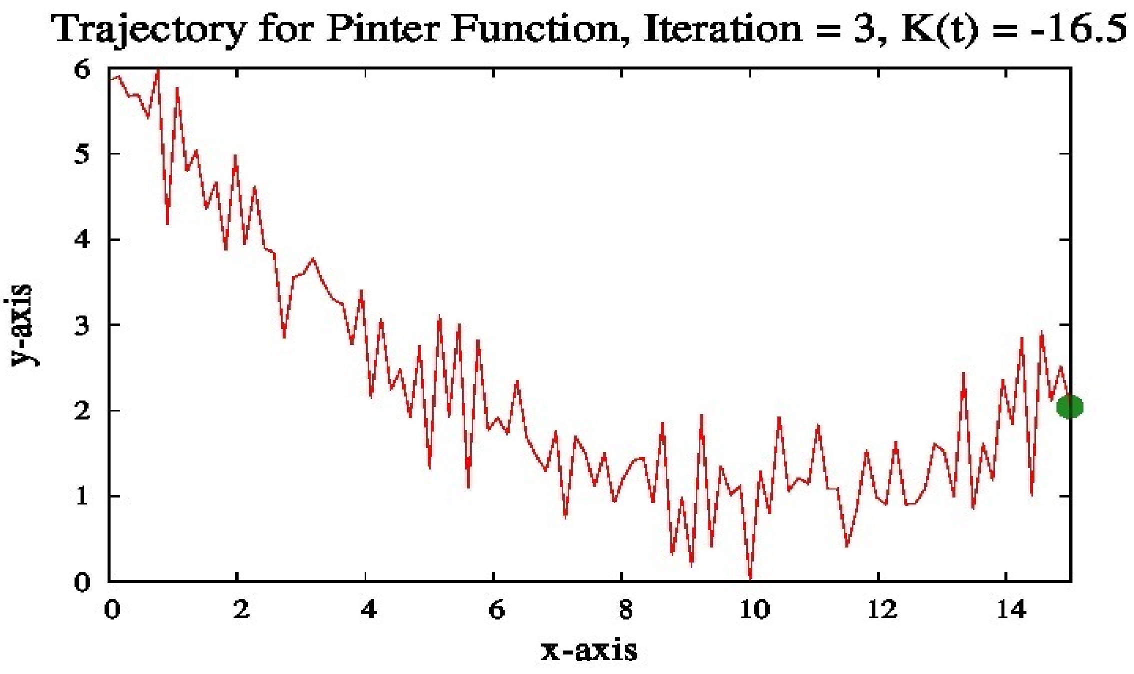

FIGURE 6.15: Algorithm Trajectory - Step 3

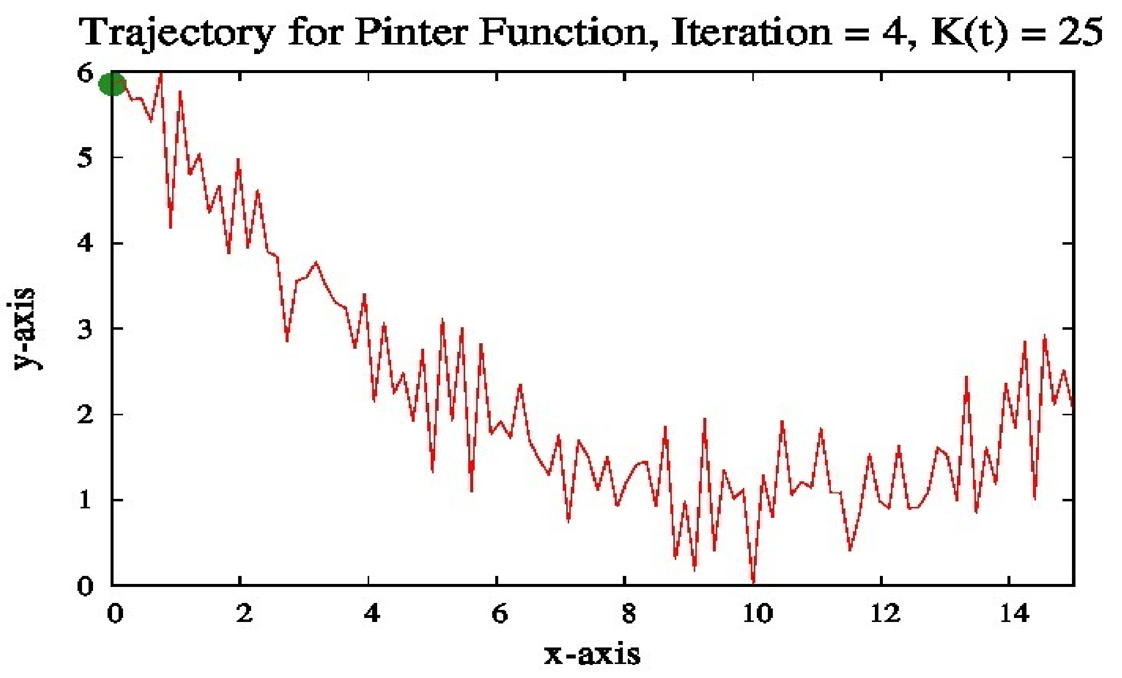

FIGURE 6.16: Algorithm Trajectory - Step 4

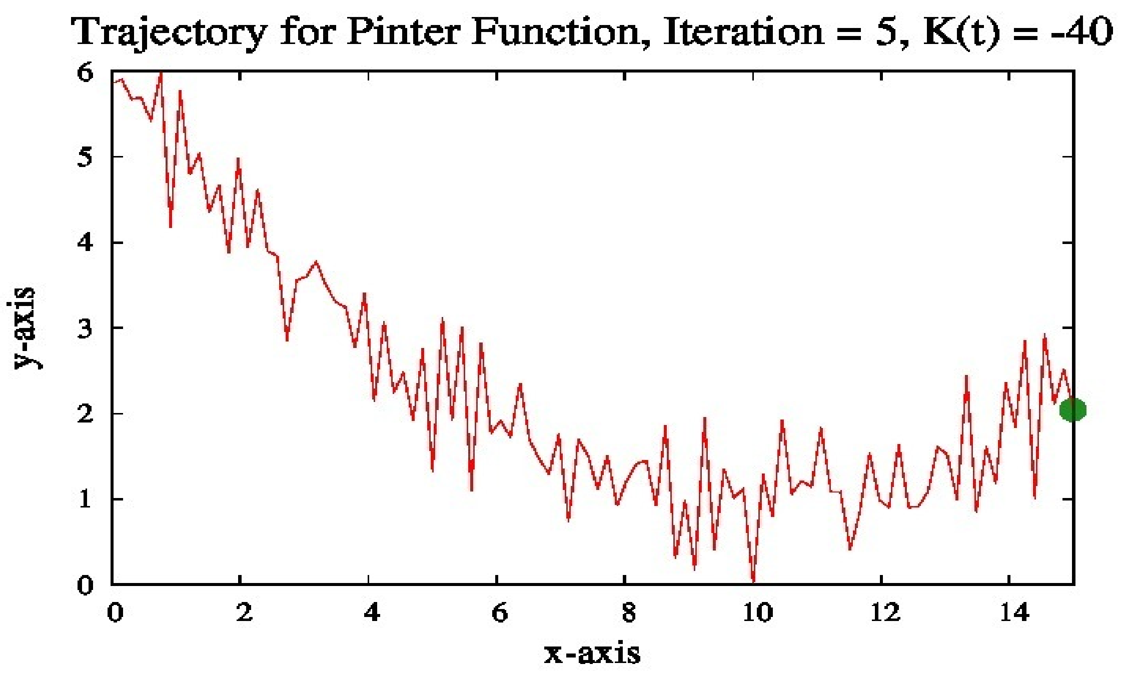

FIGURE 6.17: Algorithm Trajectory - Step 5

6.8.2.1 Trajectory Local Looping

We conclude that the trajectory gets stuck and cycle between two points: 0 and 10. Those points are boundary limits. We also observe is that as the trajectory proceeds the value of $K(t)$ gradually and considerably increases. It can be understood when the numerical integration of that term is analyzed:

$$K(t) \approx \alpha \left(\frac{k(0)}{2} + \sum_{s=1}^{n-1} k\left(s\right) + \frac{k(t)}{2} \right) \tag{6.35}$$

where:

$$k(s) = (x(t) - x(s)) \exp \left(\frac{\|x(t) - x(s)\|_2^2}{\beta} \right) \tag{6.36}$$

Hence, as the algorithm proceeds, the points are accumulated and the value of the integral term $K(t)$ grows whenever the trajectory happens to deviate from the majority of the past points. For small domains the described problem results in the trajectory 'bumping against' box constraints very quickly due to the high value of the integral term $K(t)$ and, therefore, prevents the algorithm from achieving any sensible results.

6.9 Conclusion for No Constraint Problem

As has been shown in the previous sections, the proposed method can perform very poorly in certain cases for relatively small box-constraints. Hence, we decided to test the proposed methods on the cases when there are no constraints at all. The results are taken only for Pinter function.

In the graphs below, the methods are indexed as follows:

1. - Original Version of Stochastic Gradient Descent
2. - Stochastic Gradient Descent with the integral term $K(t)$
3. - Stochastic Gradient Descent with the weighted version of the integral term $K(t)$

The first graph shows the error accuracy for all the considered algorithms. The next graph presents the number of function evaluations.

As can be observed, the number of function evaluations is roughly similar in terms of their means. However, the standard deviation of the number of function evaluations for the standard Stochastic Gradient Descent is much bigger than those for other two methods. What is more, the error accuracy for both the standard Stochastic Gradient Descent and its modified version which uses equal weights for past points are also roughly the same. However, the version of Stochastic Gradient Descent algorithm with time-weighted past points outperforms the other two versions of the algorithm.

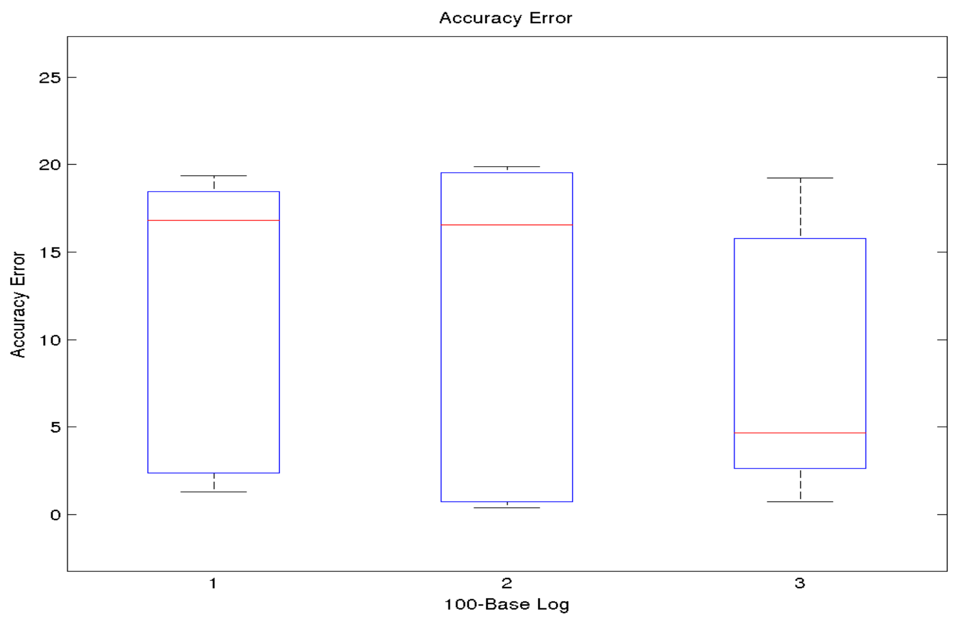

FIGURE 6.18: Pinter - Error Accuracy - No Constraints

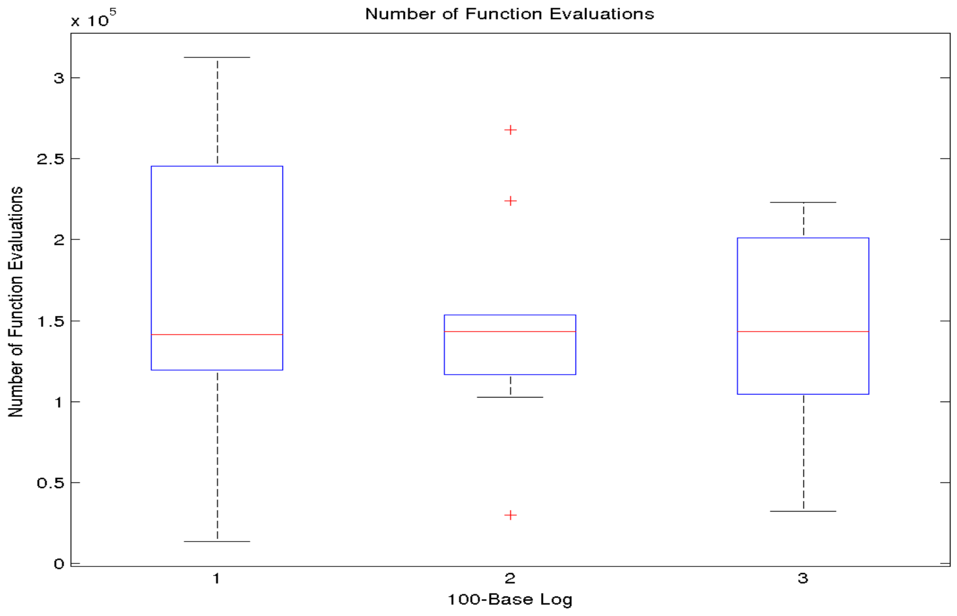

FIGURE 6.19: Pinter - Number of Function Evaluations - No Constraints

That results, that is the better performance of the version of Stochastic Gradient Descent algorithm with time-weighted past points, is quite different from the results for constrained problems. It could be explained by the fact that in unconstrained case, there are no boundaries which create the loop between a small set of points. It also shows the potential of using the past points when considering the trajectory in Stochastic Gradient Descent algorithm.

6.10 Weaknesses of Method

One of the biggest weakness of the proposed methods is the fact that they might perform extremely bad for constrained cases. What is more, whether or not the trajectory will loop between a couple of points is determined very early, within a few first iterations. Another disadvantage of the method is that we need to carefully chose the values for α, β and C (number of past points considered). However, finding the most appropriate values for those variables might be simply viewed as another optimization problem.

The method is also extremely time-consuming in comparison to the original version of the Stochastic Gradient Descent. At each iteration, we need to keep track of the last C points and we need to evaluated our $K(t)$ function. However, such an approach results in considerable slow-down. Therefore, using the proposed methods make sense, but only in the situations when objective function is also computationally hard to compute. Then, the additional cost of using the proposed methods is not considerable and could be neglected.

6.11 Final Conclusion and Future Works

We show the advantage of using the past points when considering the next evaluated point for a trajectory in the Stochastic Gradient Descent. In particular, we show that using the proposed methods might considerably improve the accuracy of the algorithm without any significant impact on the number of function evaluations. We present that using the time-weighted past points might outperform the original version of Stochastic Gradient Descent algorithm. However, that improvement comes at certain price: execution time substantially increases. We have also encountered and explained the problem of the trajectory falling in the loop of traveling between only a small number of boundary points when the proposed methods are used for problems with small domains. That problem does not seem to present when there are no constraints.

In the future, the proposed methods could be improved by finding possibly other, more appropriate values for α, β and C. However, finding those values might be just another optimization problem. What is more, the proposed methods require extremely efficient implementations so that the impact of keeping track and analyzing the past points on the overall algorithm performance is not considerable.

Chapter 7

Protein Structure Global Optimization

7.1 Problem Introduction

In this chapter, we present how our proposed techniques could be used to find out and predict what the most likely structure of proteins is.

Prediction of protein structure involves finding the tertiary structure of a protein with the least energy level, that is the one that is the most stable. Due to its chemical properties, after some time proteins will stabilize and reach its most stable state.

Protein structure prediction has many useful applications, especially in biomedicine and drug making. One of the most significant is finding out the predicted tertiary structure of proteins in drugs, which then is used to model the effect of drugs on different bacteria and viruses. Another useful application of protein structure prediction is the analysis of how different protein might react under various external conditions.

In this chapter, we choose a protein structure prediction model AB off-lattice model, described in [15] and test our proposed changes to global optimization techniques to see how successful they are for practical applications.

7.2 Protein Structure Model

Each protein structure prediction model makes different assumptions. In this dissertation, we test our proposed changes on one of simplified models. The simplified model is described in [16]. The difference between the original AB off-lattice model is dropping the constrain that the distance between neighbor proteins should be equal to 1 as described in [17].

In the model used for our tests, only two types of amino acids are used: hydrophobic, denoted by capital letter H, and hydrophilic, denote by capital letter P. It is also assumed that the only interaction is between non-adjacent, neighboring hydrophobic monomers. Despite such seemingly significant assumption, the model is still used in practice with satisfying results. More complex model, that would be more suitable for bigger problems, would also consider the forces between local neighbors and acids within monomers.

7.3 Model Details

7.3.1 2-Dimensional Problem

The 2-Dimensional Protein Structure in AB Off-lattice is expressed as:

$$E = \sum_{i=2}^{n-2} E_\theta(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{7.1}$$

where:

$$E_\theta(i) = \frac{1}{4}(1 - u_i \cdot u_{i+1}) \tag{7.2}$$

and:

$$E_{LJ}(r_{ij}, \xi_i, \xi_j) = 4\left[r_i^{-12} - C(\xi_i, \xi_j) r_{ij}^{-6}\right] \tag{7.3}$$

Finally, the constant $C(\xi_i, \xi_j)$ is defined as:

$$C(\xi_i, \xi_j) = \begin{cases} +1 & \xi_i = A \wedge \xi_j = A \\ +\frac{1}{2} & \xi_i = B \wedge \xi_j = B \\ -\frac{1}{2} & (\xi_i = A \wedge \xi_j = B) \vee (\xi_i = B \wedge \xi_j = A) \end{cases} \tag{7.4}$$

In the model expressions above, u_i denotes the position of *ith* monomer. The distance between residues i and j is denoted as r_{ij}.

In the assumed model, the objective function f is the energy functional E.

7.3.2 3-Dimensional Problem

The used protein structure 3-dimensional prediction model assumes that the energy function for any n monomers chain is expressed by:

$$E = \sum_{i=1}^{n-2} E_\theta(i) + \sum_{i=1}^{n-3} E_\tau(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{7.5}$$

where:

$$E_\theta(i) = u_i \cdot u_{i+1} \tag{7.6}$$

and:

$$E_\tau(i) = -\frac{1}{2} u_i \cdot u_{i+2} \tag{7.7}$$

$$E_{LJ}(r_{ij}, \xi_i, \xi_j) = 4C(\xi_i, \xi_j)\left[r_i^{-12} - r_{ij}^{-6}\right] \tag{7.8}$$

Finally, the constant $C(\xi_i, \xi_j)$ is defined as:

$$C(\xi_i, \xi_j) = \begin{cases} +1 & \xi_i = A \wedge \xi_j = A \\ +\frac{1}{2} & \xi_i = B \wedge \xi_j = B \\ +\frac{1}{2} & (\xi_i = A \wedge \xi_j = B) \vee (\xi_i = B \wedge \xi_j = A) \end{cases} \tag{7.9}$$

In the model expressions above, u_i denotes the position of ith monomer. The distance between residues i and j is denoted as r_{ij}.

The graph below shows the possible representation of proteins in the Cartesian coordinate system:

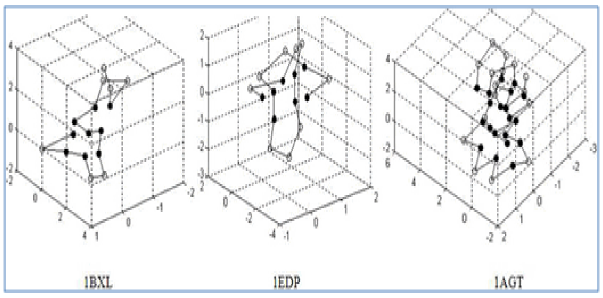

Figure 7.1: Possible representation of 3-Dimensional Protein Folding in the Cartesian coordinate system

The graph below shows the possible representation of proteins viewed as chemical monomers:

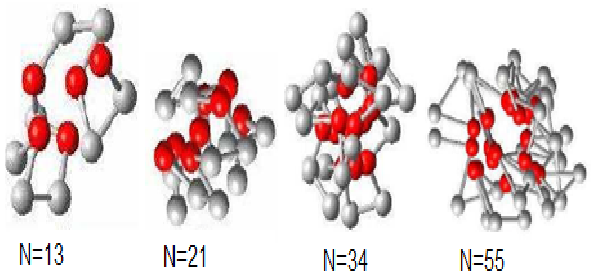

Figure 7.2: Possible representation of 3-Dimensional Protein Folding

7.4 Input Framework

Our test cases are obtained using Fibonacci sequences of proposed amino-acids. They are defined as:

$$S_0 = A \tag{7.10}$$

$$S_1 = B \tag{7.11}$$

$$S_{n+1} = S_{n-1} * S_n \tag{7.12}$$

where $*$ is a concatenation operation, that is combining amino-acids sequences of S_{n-1} and S_n. The examples are:

1. $S_0 = A$
2. $S_1 = B$
3. $S_2 = S_0 * S_1 = A * B = AB$
4. $S_3 = S_1 * S_2 = B * AB = BAB$
5. $S_4 = S_2 * S_3 = AB * BAB = ABBAB$

where the symbol A denotes hydrophobic amino-acid and the symbol B stands for hydrophilic amino-acid.

As other example, we will show how to use the function $C(\xi_i, \xi_j)$. Let us assume the amino-acid sequence is:

$$\xi = ABBAB \tag{7.13}$$

Now, the function for $i = 1$ and $j = 2$ is defined as:

$$C(\xi_1, \xi_2) = C(A, B) = -\frac{1}{2} \tag{7.14}$$

The following table presents how the structures are created:

Fibonacci Sequence Index	Fibonacci Sequence	Number of Acids Used
0	A	1
1	B	1
2	AB	2
3	BAB	3
4	ABBAB	5
5	BABABBAB	8
6	ABBABBABABBAB	13
7	BABABBABABBABBABABBAB	21

7.5 Gradient Evaluation

The gradient and its derivation of the presented AB off-lattice model is in Appendix A. In this section, we only show the final form for 2- and 3-Dimensional versions of the problem.

7.5.1 Gradient for 2-Dimensional Model

For 2-Dimensional version of the problem, we have:

$$\frac{\delta E}{\delta x_i} = \frac{\delta E_\theta(i)}{\delta x_i} + \frac{\delta E_\theta(i-1)}{\delta x_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_i} + \sum_{j=1}^{i-2} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta x_i} \tag{7.15}$$

$$\frac{\delta E}{\delta y_i} = \frac{\delta E_\theta(i)}{\delta y_i} + \frac{\delta E_\theta(i-1)}{\delta y_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_i} + \sum_{j=1}^{i-2} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta y_i} \tag{7.16}$$

for each monomer, which has the coordinate (x_i, y_i).

7.5.2 Gradient for 3-Dimensional Model

For 3-Dimensional version of the problem, we have:

$$\frac{\delta E}{\delta x_i} = \frac{\delta E_\theta(i)}{\delta x_i} + \frac{\delta E_\theta(i-1)}{\delta x_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_i} + \sum_{j=1}^{i} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta x_i} + \frac{\delta E(i)_\tau}{\delta x_i} + \frac{\delta E(i+2)_\tau}{\delta x_i} \tag{7.17}$$

$$\frac{\delta E}{\delta y_i} = \frac{\delta E_\theta(i)}{\delta y_i} + \frac{\delta E_\theta(i-1)}{\delta y_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_i} + \sum_{j=1}^{i} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta y_i} + \frac{\delta E(i)_\tau}{\delta y_i} + \frac{\delta E(i+2)_\tau}{\delta y_i} \tag{7.18}$$

$$\frac{\delta E}{\delta z_i} = \frac{\delta E_\theta(i)}{\delta z_i} + \frac{\delta E_\theta(i-1)}{\delta z_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta z_i} + \sum_{j=1}^{i} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta z_i} + \frac{\delta E(i)_\tau}{\delta z_i} + \frac{\delta E(i+2)_\tau}{\delta z_i} \tag{7.19}$$

7.6 Problem Constraints

For the described problem, we use box constrains for each dimension. The given protein is centered at the origin point of the coordinate system. For 2-dimensional problem, we have:

$$\forall i, 0 < i \le N, x_{min} \le x_i \le x_{max} \tag{7.20}$$

$$\forall i, 0 < i \le N, y_{min} \le y_i \le y_{max} \tag{7.21}$$

where N is the number of acids used.

In addition to both constraints, we have the following constraint for 3-dimensional version of the problem:

$$\forall i, 0 < i \leq N, z_{min} \leq z_i \leq z_{max} \tag{7.22}$$

7.7 Evaluation Methodology

All of the tested methods are applied to the described practical application of global optimization for finding the most optimal protein folding. The methods are numbered as:

1. Random Sampling of Starting Points (Chapter 3)
2. Sampling of Starting Points with Gaussian Processes (Chapter 3)
3. Stochastic Gradient Descent with the standard logarithmic cooling function (Chapter 4)
4. Stochastic Gradient Descent with the modified logarithmic cooling function (Chapter 4)
5. Stochastic Gradient Descent with the "decreasing to one" cooling function (Chapter 4)
6. Hyperbolic Cross Points with Non-Adaptive Point Ranking (Chapter 5)
7. Hyperbolic Cross Points with Adaptive Point Ranking (Chapter 5)
8. Hyperbolic Cross Points with Stochastic Point Ranking - Version I (Chapter 5)
9. Hyperbolic Cross Points with Stochastic Point Ranking - Version II (Chapter 5)
10. Stochastic Gradient Descent - Using Past Points Version I (Chapter 6)
11. Stochastic Gradient Descent - Using Past Points Version II (Chapter 6)

We run the simulations with different dimensionality for 2- and 3-Dimensional Problem. For each run, we take note of the total number of function evaluations, the achieved error accuracy and the execution time. We run the simulations for the proteins that contain the following acids:

1. BABABBAB
2. ABBABBABABBAB

For each configuration, we run the algorithms $N = 15$ times.

7.8 Results

7.8.1 2-Dimensional Protein Folding Results

7.8.1.1 Results for protein types: BABABBAB

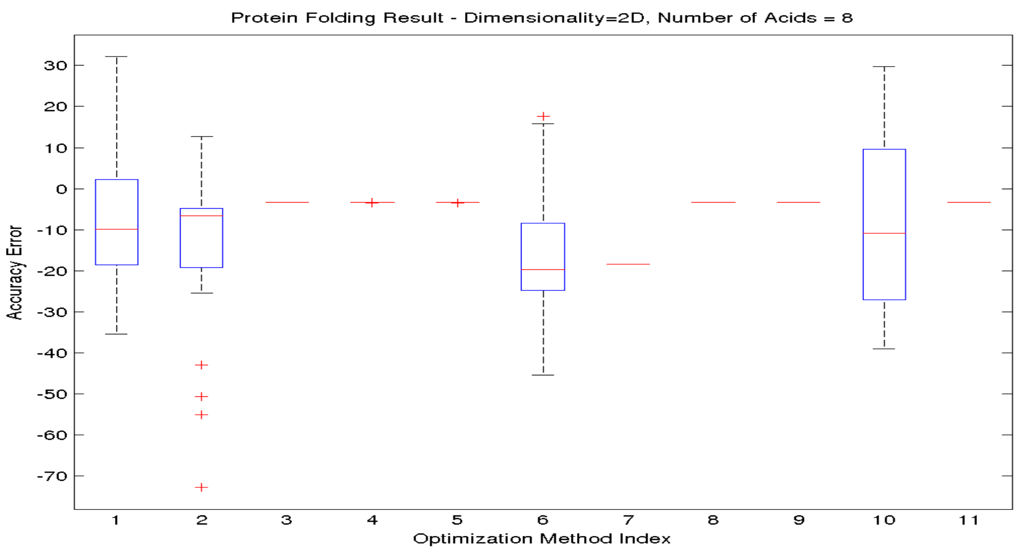

FIGURE 7.3: Protein Folding Results for 2 Dimensions and 8 proteins

7.8.1.2 Results for protein types: ABBABBABABBAB

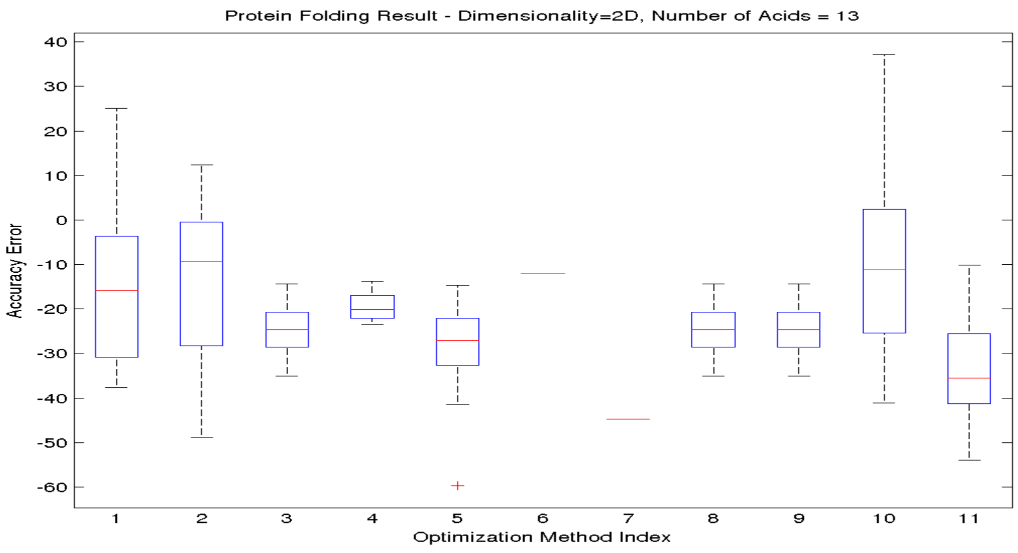

FIGURE 7.4: Protein Folding Results for 2 Dimensions and 13 proteins

7.8.2 3-Dimensional Protein Folding Results

7.8.2.1 Results for protein types: BABABBAB

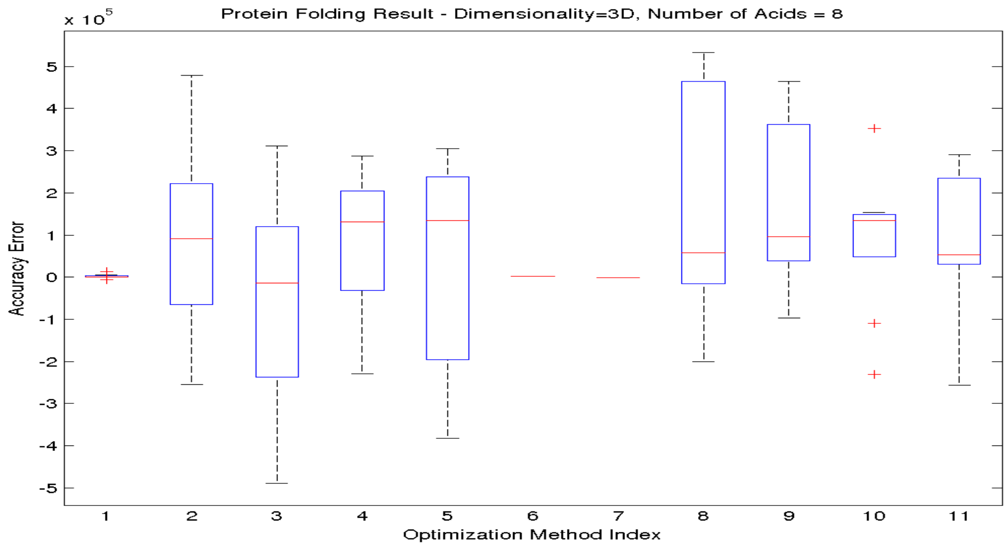

FIGURE 7.5: Protein Folding Results for 3 Dimensions and 8 proteins

7.8.2.2 Results for protein types: ABBABBABABBAB

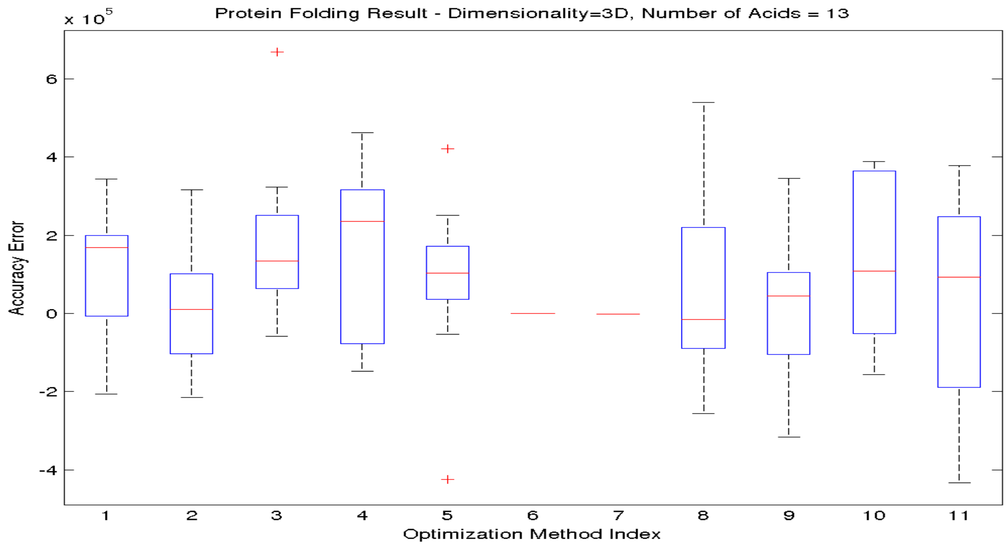

FIGURE 7.6: Protein Folding Results for 3 Dimensions and 13 proteins

7.9 Conclusion

7.9.1 Conclusions for 2-Dimensional Protein Folding

7.9.1.1 Conclusions for protein types: BABABBAB

For 2-dimensional problem with the given acid types, the graph shows that the best result has been achieved by using Sampling of Starting Points with Gaussian Processes. However, it is also a method with the greatest variance. Another, almost equally successful method is Hyperbolic Cross Points using Non-Adaptive Ranking and Stochastic Gradient Descent algorithm with added non-weighted integral term. Almost all of the other methods converged to the same solution in all of the simulations performed.

7.9.1.2 Conclusions for protein types: ABBABBABABBAB

The best result was achieved by using Stochastic Gradient Descent algorithm with added weighted integral term. Another, well-performing method was Sampling of Starting Points with Gaussian Processes. In comparison to the results for a smaller size problem, the results achieved by the methods have much bigger variance. What is more interesting, the means of the results achieved by the methods are much closer to each other than in the previous problem.

7.9.2 Conclusions for 3-Dimensional Protein Folding

7.9.2.1 Conclusions for protein types: BABABBAB

The best result, both in terms of the mean and the minimum, was achieved by the original version of Stochastic Gradient Descent algorithm. On the other hand, the worst performing method for this setting of the problem was, both in therms of the mean and the worst results, Hyperbolic Cross Points with Stochastic Point Ranking - Version I (Chapter 5). Interestingly, Random Sampling of Starting Points has hardly any variance. Both Adaptive and Non-Adaptive Hyperbolic Cross Points methods also converged to the same solution.

7.9.2.2 Conclusions for protein types: ABBABBABABBAB

For this setting of the problem, the best performing method was the Stochastic Gradient Descent Algorithm with the weighted integral term (Chapter 6). It considerably outperformed the rest of the methods. However, there is no definite worst-performing method. In terms of the mean, Stochastic Gradient Descent with the modified logarithmic cooling function (Chapter 4) achieved the worst results. However, the worst results in terms of achieving the maximum energy level was achieved by Hyperbolic Cross Points with Stochastic Point Ranking - Version I (Chapter 5).

7.9.3 Final Conclusion

In total, it seems that for the proposed model, the following two methods perform especially well:

1. Sampling of Starting Points with Gaussian Processes (Chapter 3)
2. Stochastic Gradient Descent - Using Weighted Past Points Version II (Chapter 6)

However, it is important to emphasize that there is no dominant strategy that outperform all the other methods. The two elected methods perform well in most of the simulations. However, each of the method offered some kind of trade-off, in particular some of the best performing methods did achieve a better minimal result, but at the same time they achieve much worse mean values. What is more, each of the method required different time and memory complexity. Hence, depending on the resources, some of the methods might be preferred over the others.

7.9.4 Model weaknesses

The biggest weakness of the model is the fact that it uses simplified constraints:

$$\forall i, 0 < i \leq N, x_{min} \leq x_i \leq x_{max} \tag{7.23}$$

$$\forall i, 0 < i \leq N, y_{min} \leq y_i \leq y_{max} \tag{7.24}$$

where N is the number of acids used.

In addition to both constraints, we have the following constraint for 3-dimensional version of the problem:

$$\forall i, 0 < i \leq N, z_{min} \leq z_i \leq z_{max} \tag{7.25}$$

However, in the reality those constraints cannot be clearly justified. In fact, the better approach would be to assume that the each acid within a protein is at one unit distance from each other, as described in [17] and presented below:

$$\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2} = 1 \tag{7.26}$$

In other words, the distance between consecutive particles should be equal to one.

7.10 Extended constraints

To add the proposed constraint which makes sure that the distance between consecutive particles is equal to one, we add the penalty function:

$$P(r) = \sum_{i=2}^{N} (1 - (x_i - x_{i+1})^2 - (y_i - y_{i+1})^2)^2 \tag{7.27}$$

and the energy function is now:

$$E_P(r, \lambda) = E_{LJ} + \lambda P(r) \tag{7.28}$$

The updated gradient is now for 2-dimensional problem is now:

$$\frac{\delta E_P}{\delta x_i} = \frac{\delta E_{LJ}}{\delta x_i} + 4\lambda(1-(x_i-x_{i-1})^2-(y_i-y_{i-1})^2)(x_i-x_{i-1}) - 4\lambda(1-(x_{i+1}-x_i)^2-(y_i-y_{i+1})^2)(x_{i+1}-x_i) \tag{7.29}$$

$$\frac{\delta E_P}{\delta y_i} = \frac{\delta E_{LJ}}{\delta y_i} - 4\lambda(1-(x_i-x_{i-1})^2-(y_i-y_{i-1})^2)(y_i-y_{i-1}) - 4\lambda(1-(x_{i+1}-x_i)^2-(y_i-y_{i+1})^2)(y_{i+1}-y_i) \tag{7.30}$$

Hence, setting λ to high value should result in the constraint being fulfilled and making potential results more physically accurate.

Chapter 8

Conclusion and Future Works

8.1 Project Approach Overview

The aim of the project is the analysis of back-tracking and stochastic methods used in global optimization algorithms. Back-tracking methods when used for global optimization algorithms allow us to take advantage of the already visited and analyzed past points in domain, while stochastic methods are used to make algorithms less deterministic and more robust to rich variety of different real-life problems. Back-tracking and stochastic methods analyzed in this thesis are applied to three main classes of global optimization algorithms:

1. **Multi Start Search** - running local-search deterministic algorithms from multiple starting points across domain

2. **Stochastic Gradient Descent** - solving Stochastic Differential Equation which expresses trajectory

3. **Hyperbolic Cross Points** - evaluating only points that compose Sparse Grid - approximation of multi-dimensional function that avoids Curse of Dimensionality type of complexity: $O(N^d)$, where N is the number of points in every dimensions and d is the number of dimensions

We proposed and analyzed four back-tracking and stochastic methods for Global Optimization Algorithms. The proposed methods involve:

- **Introduction of Gaussian Processes to Multi Start Search** in order to mark the regions that have already been explored and avoid re-exploring such regions

- **Introduction of Stochastic Term to Hyperbolic Cross Methods** to change the order of evaluated points, make the method less deterministic and usable for Monte-Carlo type simulations

- **Analysis and Proof of Convergence** of various cooling functions used in Stochastic Gradient Descent algorithm
- **Introduction of Integral Term to Stochastic Gradient Descent** algorithm that keeps track of the past points and decreasing the probability of re-exploring the already explored regions

8.2 Experimental Analysis Conclusions

For each of the methods, we run the experiments on three functions:

- **Pinter function**
- **Michalewicz function**
- **Restrigin function**

Each of those functions are extremely challenging for a computer to be optimized, but can be easily solved graphically by a human. The results obtained were analyzed under a few different statistics:

- **Number of function evaluations**
- **Accuracy error**
- **Execution time**
- **Number of unique evaluated points**

8.2.1 Introduction of Gaussian Processes to Multi Start Search

This method has proven to be very successful. In particular, the experimental results show that the number of function evaluations decreased for all of the tested methods. The accuracy error achieved by the method is also better than that for Random Starting Point Search. In particular, we conclude that the method works considerably well for small domains. After the closer analysis, we also conclude that the main challenge when using Gaussian Processes in Multi Start Search Algorithm is the additional computational complexity, especially when there is a huge number of local minima. One of the future works for that method could involve tackling that problem by, for instance, testing various methods of merging and setting value of standard deviations as well as efficient parallel implementation.

8.2.2 Introduction of Stochastic Term to Hyperbolic Cross Points

Another tested method involved adding Stochastic Term to Adaptive Ranking used in Hyperbolic Cross Points. After running simulations for that method and comparing the results against other ranking methods Adaptive and Non-Adaptive Rankings, we conclude that, although the proposed changes do not impact algorithm performance in terms of number of function evaluations or error accuracy, the main effect of introducing Stochastic Term to Ranking is the introduction of bigger variance in the final results. Therefore, the biggest advantage of using the proposed method is making Hyperbolic Cross Points less deterministic and usable for Monte-Carlo-type and batch simulations. The method does not impact negatively on the performance of Hyperbolic Cross Points method when compared to using Adaptive ranking, because in both cases the most computationally exhaustive part is keeping the list of candidate points sorted. Hence, we also conclude that the method could be used, given computational resources, as an alternative to Adaptive Ranking.

8.2.3 Testing different cooling functions in Stochastic Gradient Descent algorithm

In Stochastic Gradient Descent algorithm, the cooling function is used to determine the current variance of trajectory expressed by Stochastic Differential Equation. Hence, the impact of that value on the performance of the algorithm is substantial. We propose two different function and prove that the mathematical properties of Stochastic Gradient Descent algorithm such as, for example, weak convergence are maintained. The experimental results show that using each of the functions influence the number of function evaluations and error accuracy. We also conclude that every function performs differently for different function and domain sizes. Hence, they might be used to tune up the algorithm to different problems, for example, some of the methods prefer thorough scrutiny of the local basins of attraction, which makes them more useful when there is a small number of more dispersed local minimum, while others are more appropriate for exploring huge number of convexity areas. Another conclusion is that the biggest challenge in that method is matching the most appropriate cooling function to some application and future works could possibly involve creating heuristics for making that decision.

8.2.4 Introduction of Integral Term to Stochastic Gradient Descent

The last proposed method involves the introduction of the integral term that collects the information about the visited points. It is used to steer the trajectory away from the already explored regions and, mathematically, could be viewed as a bias towards the unexplored regions. We also introduce two additional variables that can be used to scale the influence of that term. Another choice that we made is about how many past points should be used for calculating the integral term, because, as it turns out, using all of the past points turns out to be computationally too challenging. The experimental results for our test functions show that the method is very promising. Although it does not lower the number of function evaluations before the algorithm terminates, it particularly improves the error accuracy and number of unique points visited in the domain. In addition to this, we also discovered that sometimes the integral term might grow too big and it causes the problem when the trajectory starts bumping against our box constraints, especially they are small. The performance for unconstrained case proves to support the usefulness of the proposed method. However, we think that the next improvement to that method could be efficient implementation of storing and processing past points as well as creating heuristics for setting the values for its option variables.

8.3 Experimental Results for Protein Folding Problem

In this dissertation, we have also run simulations for Protein Folding Problem using AB off-lattice model. Its main advantage is the concise mathematical descriptions. In addition to this, it is relatively very physically accurate. The main aim of running that application was to test the usefulness of the proposed methods as well as compare them against each other. We run a few experiments in various settings where we differ the number of proteins and dimensionality of the problem to see how it affects the performance.

The results have shown that, in general, the methods perform relatively similar and could be successfully applied in practice. There is no one dominant method that always performs better than the others. However, we found that the two methods were one of the best performing methods in all of the experimental settings:

1. **Multiple Start Search with Gaussian Processes**
2. **Stochastic Gradient Descent algorithm with added Integral Term**

8.4 Final Conclusion

After analyzing and testing the proposed back-tracking and stochastic methods, we conclude that they can be successfully used to improve the performance of the global optimization algorithms. We also think that the use of stochastic and back-tracking methods can compete against other available methods. Another conclusion from the analysis of the performance of the proposed methods is the fact that each of them offer certain trade-off. They can be tuned to particular application, but such a tuning might be also additionally challenging in terms of computational resources as it is in the case of choosing the most appropriate cooling function for Stochastic Gradient Descent or choosing the specific variables for the proposed Integral Term added to the trajectory followed by SDE Algorithm. What is more, we also note that in some methods the number of function evaluations can be compromised for the sake of improving error accuracy as in the case of adding Gaussian Processes to Multiple Start Search algorithm. Another successful outcome of the project is making Hyperbolic Cross Points method less deterministic and available for, for example, batch simulations or efficient parallel implementations. Based on the experimental results of the proposed methods, we deem the project to be successful.

8.4.1 Weaknesses and Future Works

Although the proposed methods proved to be successful, we think that there are certain trade-off one must make when using such methods. In particular, every back-tracking method is extremely computationally challenging as they usually require a lot of additional memory and CPU-time, for example, adding the Integral Term to Stochastic Gradient Descent require us to keep all or at least some part of the past points and iterating over all of them at each step. Adding Gaussian Processes also adds a lot of complexity as we need to maintain the model with our Gaussian Processes every time we choose a new point for evaluation. In the future, we think that problem could be solved using efficient parallel implementations of the methods.

Another weakness of the problem is the scope of variables that could be used to tune up the method to particular method, for example, choosing the best cooling function for Stochastic Gradient Descent algorithm. Currently, there is no heuristics for setting their values. However, we think that further experiments and broadening the scope of tested functions could result in some useful heuristics for that problem.

Appendix A

Appendix A

In this appendix, we present how we found the gradient for AB Off-Lattice model for Protein Folding problem,

A.1 2-Dimensional Problem

The 2-Dimensional Protein Structure in AB Off-lattice is expressed as:

$$E = \sum_{i=2}^{n-2} E_\theta(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{A.1}$$

where:

$$E_\theta(i) = \frac{1}{4}(1 - u_i \cdot u_{i+1}) \tag{A.2}$$

and:

$$E_{LJ}(r_{ij}, \xi_i, \xi_j) = 4 \left[r_i^{-12} - C(\xi_i, \xi_j) r_{ij}^{-6} \right] \tag{A.3}$$

Finally, the constant $C(\xi_i, \xi_j)$ is defined as:

$$C(\xi_i, \xi_j) = \begin{cases} +1 & \xi_i = A \wedge \xi_j = A \\ +\frac{1}{2} & \xi_i = B \wedge \xi_j = B \\ -\frac{1}{2} & (\xi_i = A \wedge \xi_j = B) \vee (\xi_i = B \wedge \xi_j = A) \end{cases} \tag{A.4}$$

In the model expressions above, u_i denotes the position of ith monomer. The distance between residues i and j is denoted as r_{ij}.

In the assumed model, the objective function f is the energy functional E.

A.1.1 3-Dimensional Problem

The used protein structure 3-dimensional prediction model assumes that the energy function for any n monomers chain is expressed by:

$$E = \sum_{i=1}^{n-2} E_\theta(i) + \sum_{i=1}^{n-3} E_\tau(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{A.5}$$

where:

$$E_\theta(i) = u_i \cdot u_{i+1} \tag{A.6}$$

and:

$$E_\tau(i) = -\frac{1}{2} u_i \cdot u_{i+2} \tag{A.7}$$

$$E_{LJ}(r_{ij}, \xi_i, \xi_j) = 4C(\xi_i, \xi_j) \left[r_i^{-12} - r_{ij}^{-6} \right] \tag{A.8}$$

Finally, the constant $C(\xi_i, \xi_j)$ is defined as:

$$C(\xi_i, \xi_j) = \begin{cases} +1 & \xi_i = A \wedge \xi_j = A \\ +\frac{1}{2} & \xi_i = B \wedge \xi_j = B \\ +\frac{1}{2} & (\xi_i = A \wedge \xi_j = B) \vee (\xi_i = B \wedge \xi_j = A) \end{cases} \tag{A.9}$$

In the model expressions above, u_i denotes the position of ith monomer. The distance between residues i and j is denoted as r_{ij}.

A.2 Gradient Evaluation

A.2.1 Gradient for 2-Dimensional Model

The energy function for any n monomers chain is:

$$E = \sum_{i=2}^{n-2} E_\theta(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{A.10}$$

where:

$$E_\theta(i) = \frac{1}{4}(1 - x_i x_{i+1} - y_i y_{i+1} - z_i z_{i+1}) \tag{A.11}$$

and:

$$E_{LJ}(r_{ij}, \xi_i, \xi_j) = 4 \left[r_{ij}^{-12} - C(\xi_i, \xi_j) r_{ij}^{-6} \right] \tag{A.12}$$

The distance between two monomers is expressed as:

$$r_{ij} = \|u_i - u_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{A.13}$$

We have the following gradients:

$$\frac{\delta E_\theta(i)}{\delta x_i} = -\frac{1}{4} x_{i+1} \tag{A.14}$$

$$\frac{\delta E_\theta(i)}{\delta y_i} = -\frac{1}{4} y_{i+1} \tag{A.15}$$

$$\frac{\delta E_\theta(i)}{\delta x_{i+1}} = -\frac{1}{4} x_i \tag{A.16}$$

$$\frac{\delta E_\theta(i)}{\delta y_{i+1}} = -\frac{1}{4} y_i \tag{A.17}$$

and:

$$\frac{\delta\left(\frac{1}{r_{ij}^6}\right)}{\delta x_i} = \frac{-12(x_i - x_j)}{r_{ij}^7} \tag{A.18}$$

$$\frac{\delta\left(\frac{1}{r_{ij}^{12}}\right)}{\delta x_i} = \frac{-24(x_i - x_j)}{r_{ij}^{13}} \tag{A.19}$$

$$\frac{\delta\left(\frac{1}{r_{ij}^6}\right)}{\delta x_j} = \frac{12(x_i - x_j)}{r_{ij}^7} \tag{A.20}$$

$$\frac{\delta\left(\frac{1}{r_{ij}^{12}}\right)}{\delta x_j} = \frac{24(x_i - x_j)}{r_{ij}^{13}} \tag{A.21}$$

hence:

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_i} = 4\left[\frac{-24(x_i - x_j)}{r_{ij}^{13}} - C(\xi_i, \xi_j)\frac{-12(x_i - x_j)}{r_{ij}^7}\right] \tag{A.22}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_i} = 4\left[\frac{-24(y_i - y_j)}{r_{ij}^{13}} - C(\xi_i, \xi_j)\frac{-12(y_i - y_j)}{r_{ij}^7}\right] \tag{A.23}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_j} = 4\left[\frac{24(x_i - x_j)}{r_{ij}^{13}} - C(\xi_i, \xi_j)\frac{12(x_i - x_j)}{r_{ij}^7}\right] \tag{A.24}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_j} = 4\left[\frac{24(y_i - y_j)}{r_{ij}^{13}} - C(\xi_i, \xi_j)\frac{12(y_i - y_j)}{r_{ij}^7}\right] \tag{A.25}$$

Hence, we have:

$$\frac{\delta E}{\delta x_i} = \frac{\delta E_\theta(i)}{\delta x_i} + \frac{\delta E_\theta(i-1)}{\delta x_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_i} + \sum_{j=1}^{i-2} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta x_i} \tag{A.26}$$

$$\frac{\delta E}{\delta y_i} = \frac{\delta E_\theta(i)}{\delta y_i} + \frac{\delta E_\theta(i-1)}{\delta y_i} + \sum_{j=i+2}^{n} \frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_i} + \sum_{j=1}^{i-2} \frac{\delta E_{LJ}(r_{ji}, \xi_j, \xi_i)}{\delta y_i} \tag{A.27}$$

for each monomer, which has the coordinate (x_i, y_i).

A.2.2 Gradient for 3-Dimensional Model

For 3-Dimensional Protein Folding Problem, the energy is expressed as:

$$E = \sum_{i=1}^{n-2} E_\theta(i) + \sum_{i=1}^{n-3} E_\tau(i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} E_{LJ}(r_{ij}, \xi_i, \xi_j) \tag{A.28}$$

The gradients are:

$$\frac{\delta E_\theta(i)}{\delta x_i} = x_{i+1} \tag{A.29}$$

$$\frac{\delta E_\theta(i)}{\delta y_i} = y_{i+1} \tag{A.30}$$

$$\frac{\delta E_\theta(i)}{\delta z_i} = z_{i+1} \tag{A.31}$$

$$\frac{\delta E_\theta(i)}{\delta x_{i+1}} = x_i \tag{A.32}$$

$$\frac{\delta E_\theta(i)}{\delta y_{i+1}} = y_i \tag{A.33}$$

$$\frac{\delta E_\theta(i)}{\delta z_{i+1}} = z_i \tag{A.34}$$

and:

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_i} = 4C(\xi_i, \xi_j) \left[\frac{-24(x_i - x_j)}{r_{ij}^{13}} - \frac{-12(x_i - x_j)}{r_{ij}^{7}} \right] \tag{A.35}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta y_i} = 4C(\xi_i, \xi_j) \left[\frac{-24(y_i - y_j)}{r_{ij}^{13}} - \frac{-12(y_i - y_j)}{r_{ij}^{7}} \right] \tag{A.36}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta z_i} = 4C(\xi_i, \xi_j) \left[\frac{-24(z_i - z_j)}{r_{ij}^{13}} - \frac{-12(z_i - z_j)}{r_{ij}^{7}} \right] \tag{A.37}$$

$$\frac{\delta E_{LJ}(r_{ij}, \xi_i, \xi_j)}{\delta x_j} = 4C(\xi_i, \xi_j) \left[\frac{24(x_i - x_j)}{r_{ij}^{13}} - \frac{12(x_i - x_j)}{r_{ij}^{7}} \right] \tag{A.38}$$

$$\frac{\delta E_{LJ}(r_{ij},\xi_i,\xi_j)}{\delta y_j} = 4C(\xi_i,\xi_j)\left[\frac{24(y_i-y_j)}{r_{ij}^{13}} - \frac{12(y_i-y_j)}{r_{ij}^{7}}\right] \tag{A.39}$$

$$\frac{\delta E_{LJ}(r_{ij},\xi_i,\xi_j)}{\delta z_j} = 4C(\xi_i,\xi_j)\left[\frac{24(z_i-z_j)}{r_{ij}^{13}} - \frac{12(z_i-z_j)}{r_{ij}^{7}}\right] \tag{A.40}$$

As can be noted, the gradients are the same as for 2-Dimensional Problem. Now, we need to calculate the gradient of the additional function for 3-Dimensional problem, that is:

$$E_\tau(i) = -\frac{1}{2}u_i \cdot u_{i+2} = -\frac{1}{2}\left(x_i x_{i+2} + y_i y_{i+2} + z_i z_{i+2}\right) \tag{A.41}$$

Hence, we have:

$$\frac{E_\tau(i)}{x_i} = -\frac{1}{2}x_{i+2} \tag{A.42}$$

$$\frac{E_\tau(i)}{y_i} = -\frac{1}{2}y_{i+2} \tag{A.43}$$

$$\frac{E_\tau(i)}{z_i} = -\frac{1}{2}z_{i+2} \tag{A.44}$$

and:

$$\frac{E_\tau(i)}{x_{i+2}} = -\frac{1}{2}x_{i+2} \tag{A.45}$$

$$\frac{E_\tau(i)}{y_{i+2}} = -\frac{1}{2}y_{i+2} \tag{A.46}$$

$$\frac{E_\tau(i)}{z_{i+2}} = -\frac{1}{2}z_{i+2} \tag{A.47}$$

The final gradient is:

$$\frac{\delta E}{\delta x_i} = \frac{\delta E_\theta(i)}{\delta x_i} + \frac{\delta E_\theta(i-1)}{\delta x_i} + \sum_{j=i+2}^{n}\frac{\delta E_{LJ}(r_{ij},\xi_i,\xi_j)}{\delta x_i} + \sum_{j=1}^{i}\frac{\delta E_{LJ}(r_{ji},\xi_j,\xi_i)}{\delta x_i} + \frac{\delta E(i)_\tau}{\delta x_i} + \frac{\delta E(i+2)_\tau}{\delta x_i} \tag{A.48}$$

$$\frac{\delta E}{\delta y_i} = \frac{\delta E_\theta(i)}{\delta y_i} + \frac{\delta E_\theta(i-1)}{\delta y_i} + \sum_{j=i+2}^{n}\frac{\delta E_{LJ}(r_{ij},\xi_i,\xi_j)}{\delta y_i} + \sum_{j=1}^{i}\frac{\delta E_{LJ}(r_{ji},\xi_j,\xi_i)}{\delta y_i} + \frac{\delta E(i)_\tau}{\delta y_i} + \frac{\delta E(i+2)_\tau}{\delta y_i} \tag{A.49}$$

$$\frac{\delta E}{\delta z_i} = \frac{\delta E_\theta(i)}{\delta z_i} + \frac{\delta E_\theta(i-1)}{\delta z_i} + \sum_{j=i+2}^{n}\frac{\delta E_{LJ}(r_{ij},\xi_i,\xi_j)}{\delta z_i} + \sum_{j=1}^{i}\frac{\delta E_{LJ}(r_{ji},\xi_j,\xi_i)}{\delta z_i} + \frac{\delta E(i)_\tau}{\delta z_i} + \frac{\delta E(i+2)_\tau}{\delta z_i} \tag{A.50}$$

Bibliography

[1] Panos Parpas, Berç Rustem, and Efstratios N Pistikopoulos. Linearly constrained global optimization and stochastic differential equations. *Journal of Global Optimization*, 36(2): 191–217, 2006.

[2] Erich Novak and Klaus Ritter. Global optimization using hyperbolic cross points. In *State of the art in global optimization (Princeton, NJ, 1995)*, volume 7 of *Nonconvex Optim. Appl.*, pages 19–33. Kluwer Acad. Publ., Dordrecht, 1996. doi: 10.1007/978-1-4613-3437-8_2. URL http://dx.doi.org/10.1007/978-1-4613-3437-8_2.

[3] Vincent Granville, Mirko Křivánek, and J-P Rasson. Simulated annealing: A proof of convergence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6): 652–656, 1994.

[4] Jorge J. Moré and D. C. Sorensen. Newton's method. In *Studies in numerical analysis*, volume 24 of *MAA Stud. Math.*, pages 29–82. Math. Assoc. America, Washington, DC, 1984.

[5] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[6] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Comput. J.*, 6:163–168, 1963/1964. ISSN 0010-4620.

[7] John Greenstadt. On the relative efficiencies of gradient methods. *Math. Comp.*, 21:360–367, 1967. ISSN 0025-5718.

[8] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[9] Brett Williams. *On Stochastic Differential Equations in the Ito and in the Stratonovich Sense*. PhD thesis, 2012.

[10] Francisco J Solis and Roger J-B Wets. Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30, 1981.

[11] Christian Hennig. Methods for merging Gaussian mixture components. *Adv. Data Anal. Classif.*, 4(1):3–34, 2010. ISSN 1862-5347. doi: 10.1007/s11634-010-0058-3. URL http://dx.doi.org/10.1007/s11634-010-0058-3.

[12] Y-K Hu and YP Hu. Global optimization in clustering using hyperbolic cross points. *Pattern recognition*, 40(6):1722–1733, 2007.

[13] Anatoly Zhigljavsky and Antanas Zilinskas. *Stochastic global optimization*, volume 504. Springer New York, 2007.

[14] Tzuu-Shuh Chiang, Chii-Ruey Hwang, and Shuenn Jyi Sheu. Diffusion for global optimization in rˆn. *SIAM Journal on Control and Optimization*, 25(3):737–753, 1987.

[15] Christine Kehyayan, Nashat Mansour, and Hassan Khachfe. Evolutionary algorithm for protein structure prediction. In *Advanced Computer Theory and Engineering, 2008. ICACTE'08. International Conference on*, pages 925–929. IEEE, 2008.

[16] Mao Chen and Wen-qi Huang. Heuristic algorithm for off-lattice protein folding problem. *Journal of Zhejiang University SCIENCE B*, 7(1):7–12, 2006.

[17] Marco Locatelli and Fabio Schoen. Fast global optimization of difficult lennard-jones clusters. *Computational Optimization and Applications*, 21:55–70, 2000.