ABSTRACT

DAS, PRIYAM. Derivative-Free Efficient Global Optimization of Function of Parameters from Bounded Intervals. (Under the direction of Subhashis Ghoshal, Ernest Stitzinger and Min Kang.)

In this thesis, a derivative-free algorithm for global optimization is developed for a function of parameters, each coming from a (possibly distinct) bounded interval. Main principle of this algorithm is to make jumps along the co-ordinates of the parameter one at a time with varying step-sizes within the restricted parameter space and search for the best direction to move in a greedy manner during each iteration. Unlike most of the existing methods, incorporation of parallel computing makes it even faster since the objective function is evaluated at independent directions during an update step. Requirement of parallelization grows only in the order of the dimension of the parameter space, which makes it more convenient for parallelization. A comparative study of the performances of this algorithm and other existing algorithms have been shown for a few moderate and high-dimensional benchmark global optimization problems.

Derivative-Free Efficient Global Optimization of Function of Parameters from Bounded Intervals

by
Priyam Das

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Mathematics

Raleigh, North Carolina

2016

APPROVED BY:

Subhashis Ghoshal Co-chair of Advisory Committee	Ernest Stitzinger Co-chair of Advisory Committee

Min Kang
Co-chair of Advisory Committee

DEDICATION

I would like to dedicate my whole work to my father Prasanta Kumar Das, my mother Pama Das, my brother Pramit Das, my aunt Rupali Mitra and my mentor Priyam Biswas, the people who always helped me and inspired me while going through the hardest time of my life.

BIOGRAPHY

Priyam Das is the younger son of Prasanta Kumar Das and Pama Das. He was born in Kolkata, India. He completed his 12 years of schooling in Hindu School during the period 1995–2007. After class 12, he got admission at Jadavpur University in the department of Electronics and Telecommunication engineering. After studying at Jadavpur for one year, he decided to change his line of study and got admitted to Indian Statistical Institute in 2008. He completed his bachelors and masters degrees in statistics (B.Stat and M.Stat) in 2011 and 2013 respectively. Completing the master's degree, he joined Ph.D. program in the Department of Statistics at North Carolina State University in August, 2013. The main topics of his research includes Bayesian quantile regression and it's spatial applications, non-convex constrained and unconstrained optimization techniques and Bayesian methodologies in modeling Water data of US.

Other than being a statistician, Priyam is also a great Hip-hop dancer and teacher known under the nickname "La Krusade". His dancing styles are namely popping, dubstep, liquid, roboting, krumping and tutting. He won several professional solo dance competitions and dance battles among which winning the solo dance competition 'Sanskriti – 2009' at Jadavpur University is considered as one of his best achievements of his dancing career. He also gave paid guest performances in several occasions. But unfortunately, following three surgeries related to his sports injuries, he is currently in rehabilitation with a hope to start dancing soon again.

Priyam is also a great sportsman who won 'Best sportsman award' at Indian Statistical Institute for two consecutive years in 2011 and 2012. Most of his medals were from the sports categories high jump, long jump, arm-wrestling, shot put and walking race. Priyam is a Taekwondo green belt certified by the world Taekwondo headquarter Kukkiwon. He has some training experience in Mixed martial arts as well. He also served as the skipper goalkeeper of the soccer team of Indian Statistical Institute.

Among other skills, Priyam composed several bengali songs among which "Bheshe Jay" remained quite popular among the audience. He is also a well known funny troll and meme composer, being the creator and admin of Facebook community page "ISI Trolls". Priyam can juggle upto 4 balls or rings. He is also a motivated fitness bodybuilder and he served as personal trainer to many beginner fitness bodybuilders. In the field of bodybuilding, one of his greatest achievements is squatting with 415 lbs for 3 reps. Priyam likes leg days, food and handsome amount of sleep. One of his motivating quotes which summarizes his life in a nut-shell is "INSPIRE or DIE".

ACKNOWLEDGEMENTS

First, I would like to acknowledge Dr. Subhashis Ghoshal, my Statistics Ph.D. advisor. I came up with this algorithm while working on one of the challenging optimization problems proposed by him. I would really like to thank Dr. Rudrodip Majumdar for helping me correcting the earlier versions of this draft. I appreciate the help of my colleague Debraj Das who helped me correcting the theoretical properties in the earlier versions. I would also like to thank Suman Chakraborty for his valuable suggestions for improving the quality of this paper in the early stage. Last but not the least, I would like to thank Dr. Hua Zhou for enriching my computational skills which helped me a lot in coming up with this article.

I am thankful to my best buddy Longshaokan Wang for helping me everytime I needed that badly. Besides, I am grateful to my friends and colleagues Srinjoy Chattopadhyay, Arnab Hazra, Sohini Raha, Moumita Chakraborty, Indranil Sahoo, Arkaprava Roy, Arunava Nag, Sujatro Chakladar, Arkopal Choudhury, Sayan Patra, Poulami Maitra, Mityl Biswas, Abhishek Banerjee, Arnab Bose, Ajay Kumar, Arnab Chakraborty, Kiran Nihlani, Debashis Chatterjee. Among my seniors, I am thankful to Rajarshi Das Bhowmik, Sayantan Banerjee, Prithwish Bhaumik, Souvik Chandra, Saswata Sahoo, Samarpan Majumder, Rinku Majumder. Among my distant friends, I am really glad to Sayan Roy, Deborshee Sen, Raka Mandal, Atin Ghosh for keeping in touch with me so regularly.

My heartiest thanks goes to Dr. Shuva Gupta, Abdullah Mughrabi, Mike Chang for motivating me into the world of fitness and bodybuilding. I would like to thank Eminem, Lil Wayne, Nicki Minaj, Rupam Islam among the greatest music artists of my inspiration. Among the dancers, I would like to thank my biggest inspirations Robert Muraine, Les Twins, Chadd Smith, Tushar Kapoor, Harshvardhan Bhan, Anurag Ghosh. I am also grateful to my Taekwondo instructor Master Pradipta Kumar Roy, Master Ruma Roy Chowdhury and my soccer coach Trijit Das. Among the other helpful hands and inspirations I must thank Dr. Sourabh Bhattacharya and Dr. Arunava Goswami for helping me in the process of application to this university from my country, India.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1

Introduction

Suppose we have an objective function f of the parameters $x_1, \cdots, x_n$ coming from $S \in \mathbb{R}^n$ such that

$$f : S \mapsto \mathbb{R}$$

for $S = \prod_{j=1}^{n} I_j$ where $I_j = [a_j, b_j]$ are closed and bounded intervals on $\mathbb{R}$ for $i = 1, \cdots, n$. We want to minimize f on the given domain S. For convex objective function, since domain S is convex, convex optimization algorithms can be used for global optimization. Many efficient algorithms have been proposed in this setup. 'Gradient descent (GD)' method (see [15]) is quite efficient in solving convex optimization problems. In case the closed form derivative of the objective function is not available, numerical approximation of the derivative can be calculated and used for optimization using GD method. However, for some complex functions, finding the derivative numerically might be time consuming which might affect the efficiency of this algorithm.

'Trust Region Reflective (TRF)' algorithm (see [5], [4], [6]), 'Interior-point (IP)' algorithm (see [18], [12], [3], [20]) and 'Sequential Quadratic Programming (SQP)' algorithm (see [17], [20], [2]) are also widely used for convex optimization problems. But the main problem with these 4 aforementioned algorithms is they look for local solution and hence they are more likely get struck at any local minima in case the objective function has multiple minimas. So a good starting point is necessary if we want to optimize a non-convex function using any of these methods. One possible solution to avoid this problem is to start the iterations from several starting points. Although, for low dimensional non-convex optimization problems, the strategy of starting from multiple initial points might be affordable, but with increasing dimension of the parameter space, this strategy proves to be computationally very expensive since with increasing dimension, the requirement of number of starting points increases exponentially.

In many types of existing optimization algorithms, the main motivation is to minimize the number of function evaluations required to find a reasonable local minimum (e.g., [10]) which is not desirable if finding the global minimum is our main objective. In last few decades, many non-convex global optimization strategies have been proposed. The 'Genetic algorithm (GA)' (see [7], [1], [9]) and 'Simulated annealing (SA)' (see [14], [11]) are quite popular among them and are widely used. Although GA performs well for lower dimensional problems, GA does not scale well with complexity because in higher dimensional optimization problems there is often an exponential increase in search space size (see [8], page 21). Besides, one major problem with these two afore-mentioned methods is they might be much expensive in case we use these methods for optimizing simple convex functions without the knowledge of it's convexity.

In small dimensional parameter space, within an iteration, making an update in 'not so important' direction should not affect as much. But it is intuitive that for higher dimensional problems, it is important to move in the best possible direction during updating the parameter within an iteration. In most of the existing methods, either it finds out the best possible direction of the movement by evaluating the derivative (e.g., GD method) or it randomly selects any coordinate of the parameter and makes a jump of adaptive step size (e.g., [13]). Though derivative-based methods work pretty fast and well for some smooth objective functions, but, as mentioned earlier, in case a closed form of the derivative does not exist, numerical approximation of derivative might affect the computational efficiency. Secondly, the derivative based methods (e.g., GD, IP, SQP) mostly look for local solution. The strategy of choosing a parameter randomly and moving with step size (may be adaptive) depending on it's effect on the objective function (e.g., [13]) works well for low-dimensional problems, but in a high-dimensional setup, choosing the co-ordinates of the parameter randomly before movement decreases the chance of the moving in the best possible direction during iterations. This motivates us to find a novel way for selecting the best possible direction of movement during each step avoiding derivative and choosing the coordinate of the parameter for movement in a greedy manner instead of choosing it randomly.

In order to avoid derivatives, in an iteration, we make jumps along the co-ordinates of the parameter one at a time with varying step sizes within the parameter space and search for the best direction of movement. During an iteration, once we fix the step size, say s, for each parameter, there are two available options for movement, either in positive or negative direction by that pre-determined step size s. So there are $2n$ possibilities of of movement since we are moving only one parameter at a time by that pre-determined step-size s. Before deciding the final movement, we evaluate our objective function at these $2n$ locations and choose the best

direction to move. Thus we can ensure the best possible direction of movement at each step by greedy coordinate descent. Another advantageous side of our algorithm is that we can evaluate the objective function parallely in $2n$ directions in parallel. The requirement of parallel computing only increases in the order of the number of parameters, unlike the way Monte Carlo uses parallel computing. Because as mentioned in [13], in Monte Carlo, with increasing dimension, sampling rates should be increased exponentially but it can be only increased linearly using parallel computing. We call this algorithm 'Greedy Coordinate Descent of Varying Step-size on Bounded Intervals' (GCDVSBI).

Chapter 2

Algorithm and theoretical properties

In this chapter we propose a new algorithm for global optimization of any non-convex function whose parameter space is given by a hyper-rectangle. We also show the theoretical justification of the fact that the proposed algorithm returns a global minimum while minimizing a convex function.

2.1 Algorithm

Suppose we have a objective function $Y = f(\boldsymbol{x})$ where $\mathbf{x} = (x_1, \cdots, x_n)$ is the parameter of dimension n. Our objective is to

$$\begin{aligned} &minimize \ : f(\mathbf{x}) \\ &subject\ to \ : \mathbf{x} \in S \subset \mathrm{R}^n \end{aligned} \tag{2.1}$$

where $S = \prod_{j=1}^{n} I_j$, $I_j = [a_j, b_j]$ are closed and bounded intervals on R for $i = 1, \cdots, n$. Now consider the bijection

$$g : S \mapsto [0,1]^n \tag{2.2}$$

where $g(\mathbf{z}) = (g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})) \in [0,1]^n$ is such that $g_i(\mathbf{z}) = \frac{z_i - a_i}{b_i - a_i}$. So, without loss of generality, we can assume the domain of $\mathbf{x}$ to be $S = [0,1]^n$.

Our algorithm consists of several *runs*. Each *run* is an iterative procedure and a *run* stops based on some convergence criteria (see the algorithm). At the end of each *run* a solution is returned. Suppose we start our first *run* from any point $\boldsymbol{x}^{(1)} = (x_1^{(1)}, \cdots, x_n^{(1)})$ within the domain S. After the first *run*, following *runs* will start from the solution obtained by the *run* just before it. For example, 4^{th} *run* will start from the solution obtained by 3^{rd} *run*. So it is

required to set the starting point only for the first *run*. Each run tries to improve the solution in a 'greedy' manner by making coordinate-wise jumps through a sequence of decreasing step sizes gradually decreasing to zero, around the solution obtained by the previous *run* (see below for details). Thus, with each *run*, the solution either gets improved or remains unchanged. If two consecutive *runs* give the same solution, our algorithm stops.

In our algorithm, each *run* is similar except the values of the tuning parameters which can be reset after each *run*. In a typical *run*, we have three tuning parameters which are initial *global step size* $s_{initial}$, *step decay rate* ρ (It is either equal to ρ_1 and ρ_2, see below for details), *step size threshold* ϕ respectively. For the first *run*, we set $\rho = \rho_1$ and for following *runs*, we set $\rho = \rho_2$. Other tuning parameters are kept same of all the *runs*. In every iteration we have a parameter called *global step size* (denoted by $s^{(j)}$ for j-th iteration) and $2n$ local parameters called *local step sizes* (denoted by $\{s_i^+\}_{i=1}^n$ and $\{s_i^-\}_{i=1}^n$). In a *run*, we start the first iteration setting the *global step size* $s^{(1)} = s_{initial}$. Within every iteration, it's value is kept unchanged throughout all the operations. But at the end of the iteration, based on some criteria (see step (6) of **STAGE 1**) either it is kept same or decreased by a factor of ρ. The next iteration is started with that new value of *global step size*. So the value of $s^{(j+1)}$ can be either $s^{(j)}$ or $\frac{s^{(j)}}{\rho}$. In an iteration, the local parameters $\{s_i^+\}_{i=1}^n$ and $\{s_i^-\}_{i=1}^n$ keep updating themselves and finally they become the step sizes of the movement of the positions $i = 1, \cdots, n$ of the parameter $\mathbf{x}$ in positive and negative directions respectively. At the beginning of any iteration, we set the values of all these *local step sizes* to be equal to the *global step size* of that particular iteration, i.e., at the beginning of jth iteration, we set, $s_i^+ = s_i^- = s^{(j)}$ for $i = 1, \cdots, n$. Suppose we are moving in positive direction of i-th coordinate of $\mathbf{x}$. Assume the current value of $\mathbf{x}$ at the j-th iteration is $\mathbf{x}^{(j)} = (x_1^{(j)}, \ldots, x_n^{(j)})$. If moving $x_i^{(j)}$ by s_i^+ (whose initial value is equal to the *global step size* of that particular iteration) in the positive direction generates a point outside the domain, then s_i^+ is divided by the factor ρ (i.e., setting $s_i^+ = \frac{s_i^+}{\rho}$) and checked whether $(x_i^{(j)} + s_i^+)$ (with updated s_i^+) is still within the domain or not. In this way, s_i^+ is decreased by a division factor of ρ until $(x_i^{(j)} + s_i^+)$ is in the domain of x_i (or in this case, less than 1 since $x_i \in [0, 1]$, $s_i^+ > 0$ and it is ensured that $x_i^{(j)} \in [0, 1]$). Once the value of s_i^+ becomes less than *threshold parameter* ϕ we stop updating that particular s_i^+ and start working with the *local step size* of the next move (e.g., s_i^-, $s_{(i+1)}^+$ etc.). So in short, in an iteration, there will be always only one *global step size* and $2n$ *local step sizes* which are initialized within the iteration being equal to *global step size* and at the end of the iteration, each of them end up being less than or equal to the *global step size* of that iteration. Hence in a *run*, the *global step size* might decrease or remain same after each iteration. On the other hand, the *local step sizes* have memory-less properties since their values do not depend of their old values in the previous iteration. A *run* ends when *global step size* becomes smaller than ϕ.

step decay rate ρ determines the rate of change of global step-sizes at the end of each iteration as well as the rate of change of *local step sizes* inside an iteration. The value of ρ must be greater than 1. Taking smaller values of ρ will make the decay of step sizes slower, which would allow finer search within the domain at the cost of more computation time. Once we get a solution from first *run* setting $\rho = \rho_1$, to increase the likelihood of finding the global solution, we again start following *runs* with smaller decay rate ρ_2. Although a reasonable range for ρ_1 and ρ_2 are $[1.05, 3]$ and $[1.01, \rho_1]$ respectively, but from our experience it is noted that $\rho_1 = 2, \rho_2 = 1.05$ works well for most of the cases. The *step size threshold* ϕ controls the precision. This is the minimum possible value *global step size* can take. Once the *global step size* goes below ϕ, the *run* stops. Setting the value of ϕ to be smaller results in better precision in the cost of higher computation time. The default value of *step size threshold* ϕ is taken to be 10^{-5}. In case more precision is required, or if there is knowledge of possibility of multiple local minimas within a very small neighborhood, ϕ can be taken to be smaller. **max_iter** denotes the maximum number of iterations under each *run* and **max_runs** denotes the maximum number of allowed *runs*. **tol_fun** determines the cut-off amount of improvement of the objective function required in a single iteration to keep the *global step size* unchanged. As default we take $\textbf{max\_iter} = 50000, \textbf{max\_runs} = 1000$ and $\textbf{tol\_fun} = 10^{-15}$. Before going through **STAGE 1** for the very first time, we set $R = 1, \rho = \rho_1$ and initial guess of the solution $\mathbf{x}^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)})$.

STAGE : 1

1. Set $j = 1$. Set $s^{(j)} = s_{initial}$ Go to step (2).
2. If $j >$ **max_iter**, set $\hat{\boldsymbol{x}} = \boldsymbol{x}^{(j-1)}$, go to step (8). Else, set $s_i^+ = s_i^- = s^{(j)}$ and $f_i^+ = f_i^- = Y^{(j)} = f(\boldsymbol{x}^{(j)})$ for all $i = 1, \cdots, n$. Set $i = 1$ and go to step (3).
3. If $i > m$, set $i = 1$ and go to step (4). Else, go to step (3.1).
   (a) If $s_i^+ \leq \phi$, set $i = i + 1$ and go to step (3). Else (if $s_i^+ > \phi$), evaluate vector $\boldsymbol{q}_i^+ = (q_{i1}^+, \cdots, q_{in}^+)$ such that
   $$\begin{aligned} q_{il}^+ &= x_i^{(l)} + s_i^+ && \text{for} \quad l = i \\ &= x_i^{(l)} \qquad \text{otherwise} \end{aligned}$$
   Go to step (3.2).
   (b) Check whether $q_{ii}^+ \leq 1$ or not. If $q_{ii}^+ \leq 1$, go to step (3.3). Else, set $s_i^+ = \frac{s_i^+}{\rho}$ and go to step (3.1)
   (c) Evaluate $f_i^+ = f(\boldsymbol{q}_i^+)$. Set $i = i + 1$ and go to step (3).

4. If $i > n$, go to step (5). Else, go to step (4.1).

   (a) If $s_i^- \leq \phi$, set $i = i + 1$ and go to step (4). Else (if $s_i^- > \phi$), evaluate vector $\boldsymbol{q}_i^- = (q_{i1}^-, \cdots, q_{in}^-)$ such that

   $$\begin{aligned} q_{il}^- &= x_i^{(l)} - s_i^- \quad \text{for} \quad l = i \\ &= x_i^{(l)} \qquad \text{otherwise} \end{aligned}$$

   Go to step (4.2)

   (b) Check whether $q_{ii}^- \geq 0$ or not. If $q_{ii}^- \geq 0$, go to step (4.3). Else, set $s_i^- = \frac{s_i^-}{\rho}$ and go to step (4.1)

   (c) Evaluate $f_i^- = f(\boldsymbol{q}_i^-)$. Set $i = i + 1$ and go to step (4).

5. Set $k_1 = \underset{1 \leq l \leq m}{\arg\min}\ f_l^+$ and $k_2 = \underset{1 \leq l \leq m}{\arg\min}\ f_l^-$. If $\min(f_{k_1}^+, f_{k_2}^-) < Y^{(j)}$, go to step (5.1). Else, set $\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)}$ and $Y^{(j+1)} = Y^{(j)}$, set $j = j + 1$. Go to step (6).

   (a) If $f_{k_1}^+ < f_{k_2}^-$, set $\boldsymbol{x}^{(j+1)} = \boldsymbol{q}_{\boldsymbol{k_1}}^+$, else (if $f_{k_1}^+ \geq f_{k_2}^-$), set $\boldsymbol{x}^{(j+1)} = \boldsymbol{q}_{\boldsymbol{k_2}}^-$. Set $j = j + 1$. Go to step (6).

6. If $\sum_{i=1}^{n} (\boldsymbol{x}^{(j)}(i) - \boldsymbol{x}^{(j-1)}(i))^2 < \textbf{tol\_fun}$, set $s^{(j)} = s^{(j-1)}/\rho$. Go to step (7). Else, set $s^{(j)} = s^{(j-1)}$. Go to step (2).

7. If $s^{(j)} \leq \phi$, set $\hat{\boldsymbol{x}} = \boldsymbol{x}^{(j)}$. Go to step (8).

8. **STOP** execution. Set $\mathbf{z}^{(R)} = \hat{\mathbf{x}}$. Set $R = R + 1$. Go to **STAGE 2**.

STAGE : 2

1. If $\mathbf{R} > \textbf{max\_runs}$ or $\mathbf{z^{(R)} = z^{(R-1)}}$, **STOP** and **EXIT**. Else go to step (2)

2. Set $\rho = \rho_2$ keeping other tuning parameters (ϕ and $s_{initial}$) intact. Repeat algorithm described in **STAGE 1** setting $\mathbf{x}^{(1)} = \mathbf{z^{(R-1)}}$. If Else repeat step (1).

Setting *step size threshold* to a sufficiently small value, it can be shown that if the function is convex and all the partial derivatives exist at the solution after first *run*, the objective function value at the solution is the global minimum (see Section 2.3). However for non-convex functions, there is no way to ensure whether that is a global minimum or not. To increase the likelihood of reaching the global minimum, the **STAGE 1** of the algorithm is repeated starting from the solution obtained from the last *run* until the solution returned by two consecutive *runs* are same. In the first *run*, we set $\rho = \rho_1$ and for the following runs, we set $\rho = \rho_2$. After first *run*, setting smaller value of ρ results in slower decay of step size in the iterations of the following

runs. Thus, we look for better solution in the domain changing each co-ordinate one at a time for a sequence of finite step sizes slowly decaying to zero.

2.2 Order of algorithm

Suppose a objective function f of n variables $\mathbf{x} = (x_1, \ldots, x_n)$ is to be minimized. At the beginning of each iteration, we initialize zero arrays $\mathbf{s}^+, \mathbf{s}^-, \mathbf{f}^+$ and $\mathbf{f}^-$ where each of them is of each of size n (see step (2) of **STAGE 1** section 2.1). During each iteration, we evaluate the function at $2n$ points obtained by increasing or decreasing one parameter at a time by corresponding steps-sizes. Since there is a constraint that each parameter x_i should belong to $[0, 1]$ for $i = 1, \cdots, n$, we might need to update the step size for each position of the vector individually. Consider the update step of the i-th position of the vector $\mathbf{x}^{(j)}$ where j denotes the iteration number. The movement of that coordinate can be either in positive or negative direction. For the time being, consider the maximum number of basic operations required for positive movement only. Suppose *step size threshold* is ϕ, *parameter threshold* is λ and the *step decay rate* is ρ. The maximum step size is the initial step size $s_{initial}$. In step (3.1) and (3.2) of **STAGE 1** in Section 2.1, for some $i \in \{1, \ldots, n\}$, suppose s_i^+ is updated atmost k times. So, we have $\frac{s_{initial}}{\rho^{k-1}} \leq \phi$ and $\frac{s_{initial}}{\rho^{k-2}} > \phi$. Hence $k = 1 + \left[\frac{\log(\frac{s_{initial}}{\phi})}{\log(\rho)}\right]$ where $[z]$ returns the largest integer less than or equal to z. After each update of s_i^+, the value of step-size is compared with ϕ. In case, it is less than or equal to ϕ, no update is done at i-th position and the vector $\mathbf{q}_i^+$ is kept same as $\mathbf{x}^{(j)}$. If it is greater than ϕ, we add the updated step-size with $x_i^{(j)}$ and check whether that is greater than 1 or not. If it is greater than 1, we again update s_i^+, otherwise, we evaluate the function at the updated $\mathbf{x}^{(j)}$, i.e., $\mathbf{q}_i^+$. So, during the positive update of any given position i of the vector $\mathbf{x}^{(j)}$ in j-th iteration, maximum number of possible values s_i^+ can take is k and for each value of k, we need atmost 4 operations in the worst case scenario (in terms of the number of operations required) which are (a) adding s_i^+ with $x_i^{(j)}$, (b) checking whether $x_i^{(j)} > 1$ (c) setting $s_i^+ = \frac{s_i^+}{\rho}$ (assuming $x_i^{(j)} > 1$) and (d) checking $s_i^+ > \phi$ or not. Hence, during an iteration, the positive update of a position requires atmost $4k$ basic operations. Similarly it can be shown that for negative update of a position also, it would require atmost $4k$ operations in an iteration. Since, the update is done at n sites and each site has positive as well as negative updates, the number of operations required for update (in step (3) and (4) of **STAGE 1** in Section 2.1) in an iteration is not more than $2 * n * (4k) = 8kn$.

After evaluating the function values at $2n$ sites given by $\{f_i^+\}_{i=1}^n$ and $\{f_i^-\}_{i=1}^n$, we find the maximum of these $2n$ values and then we compare the maximum of $\{f_i^+\}_{i=1}^n$ and $\{f_i^-\}_{i=1}^n$ with current value of the function $Y^{(j)} = f(\mathbf{x}^{(j)})$ (see step (5) of **STAGE 1** in Section 2.1). So overall, it takes $2n$ operations to find the maximum of $2n + 1$ quantities. To find the square of

euclidean distance between $\mathbf{x}^{(j)}$ and $\mathbf{x}^{(j+1)}$ (see step (6) of **STAGE 1** in Section 2.1), it takes not more than $(3n-1)$ operations, i.e., n operations to compute the coordinate wise distances, n operations to square each of them and $(n-1)$ operations to find the sum of n squared coordinate wise distances. Hereby, for each iteration, the maximum number of operations required is not more than $8kn + 2n + 3n = (8k+5)n$.

The number of times the objective function is evaluated at each iteration is $2n$ (n times at each step (3.3) and (4.3) of **STAGE 1** in Section 2.1). Thus our algorithm has time complexity of $\mathbf{O}(n)$ both in terms of the number of required basic operations and number of function evaluations in each iteration step.

2.3 Theoritical properties

In this section we show that under some regularity conditions, our algorithm gives the global minimum of the objective function when it it convex. Consider the following theorem

Theorem 1. *Suppose $f : [0,1]^n \mapsto \mathbb{R}$ is a convex function. Suppose, all partial derivatives of f exist at a point $\mathbf{u} = (u_1, \ldots, u_n) \in [0,1]^n$. Consider a sequence $\delta_k = \frac{s}{\rho^k}$ for $k \in \mathrm{N}$ and $s > 0, \rho > 1$. Define $\mathbf{u}_k^{(i+)} = (u_1, \ldots, u_{i-1}, u_i + \delta_k, u_{i+1}, \ldots, u_n)$ and $\mathbf{u}_k^{(i-)} = (u_1, \ldots, u_{i-1}, u_i - \delta_k, u_{i+1}, \ldots, u_n)$ for $i = 1, \cdots, n$. If for all $k \in \mathrm{N}$, $f(\mathbf{u}) \leq f(\mathbf{u}_k^{(i+)})$ and $f(\mathbf{u}) \leq f(\mathbf{u}_k^{(i-)})$ for all $i = 1, \cdots, n$, the global minimum of f occurs at $\mathbf{u}$.*

Proof of Theorem 1. Take an open neighborhood $\mathbf{U} \subset [0,1]^n$ w.r.t. l_∞-norm containing $\mathbf{u}$ at the center. So, there exists $r > 0$ such that $\mathbf{U} = \prod_{i=1}^n U_i$ where $U_i = (u_i - r, u_i + r)$ for $i = 1, \ldots, n$. For some $i \in \{1, \ldots, n\}$, define $g_i : U_i \mapsto \mathrm{R}$ such that $g_i(z) = f(u_1, \ldots, u_{i-1}, z, u_{i+1}, \ldots u_n)$. Since f is convex on $\mathbf{U}$, it can be easily shown that g_i is also convex on U_i. We claim that $g_i(u_i) \leq g_i(z)$ for all $z \in U_i$.

Suppose there exist a point $u_i^* \in U_i$ such that $g_i(u_i^*) < g_i(u_i)$. Take $d = |u_i^* - u_i|$. Clearly $0 < d < r$. Without loss of generality, assume $u_i^* > u_i$. Hence $u_i^* = u_i + d$. Since δ_k is a strictly decreasing sequence going to 0, there exists a N such that for all $k \geq N$, $\delta_k < d$. Now we have $u_i < u_i + \delta_N < u_i + d$. Now there exists a $\lambda \in (0,1)$ such that $u_i + \delta_N = \lambda u_i + (1-\lambda)(u_i + d)$.

From convexity of g_i, we have

$$\begin{aligned} f(u_i + \delta_N) &= f(\lambda u_i + (1-\lambda)(u_i + d)) \\ &\leq \lambda f(u_i) + (1-\lambda) f(u_i + d) \\ &= \lambda f(u_i) + (1-\lambda) f(u_i^*) \\ &= \lambda f(u_i) + (1-\lambda) f(u_i) + (1-\lambda)(f(u_i^*) - f(u_i)) \\ &= f(u_i) - (1-\lambda)(f(u_i) - f(u_i^*)) \\ &< f(u_i) \quad (\text{since } f(u_i^*) < f(u_i)). \end{aligned}$$

But, we know $f(u_i) \leq f(u_i + \delta_N)$. Hence it is a contradiction.

Since partial derivatives of f exist at $\mathbf{x} = \mathbf{u}$, g_i is differentiable at $z = u_i$. Since u_i is a local minima of g_i in U_i, we have $g_i'(u_i) = 0$. So we have $\frac{\partial}{\partial x_i} f(\mathbf{x})|_{\mathbf{x}=\mathbf{u}} = 0$. By similar argument it can be shown that $\frac{\partial}{\partial x_j} f(\mathbf{x})|_{\mathbf{x}=\mathbf{u}} = 0$ for all $j = 1, \ldots, n$. Since f is convex and $\nabla f(\mathbf{u}) = 0$, $\mathbf{u}$ is a local minima in $\mathbf{U}$. Now, since $\mathbf{U} \subset [0,1]^n$, $\mathbf{u}$ is also a local minima of $[0,1]^n$. But f is convex on $[0,1]^n$. Since any local minimum of a convex function is necessarily global minimum, the global minimum of f occurs at $\mathbf{u}$. □

Corollary 1. *Suppose $f : [0,1]^n \mapsto \mathrm{R}$ is convex and differentiable function. Consider a sequence $\delta_k = \frac{s}{\rho^k}$ for $k \in \mathrm{N}$ and $s > 0, \rho > 1$. Define $\mathbf{u}_k^{(i+)} = (u_1, \ldots, u_{i-1}, u_i + \delta_k, u_{i+1}, \ldots, u_n)$ and $\mathbf{u}_k^{(i-)} = (u_1, \ldots, u_{i-1}, u_i - \delta_k, u_{i+1}, \ldots, u_n)$ for $i = 1, \cdots, n$. If for all $k \in \mathrm{N}$, $f(\mathbf{u}) \leq f(\mathbf{u}_k^{(i+)})$ and $f(\mathbf{u}) \leq f(\mathbf{u}_k^{(i-)})$ for all $i = 1, \cdots, n$, the global minimum of f occurs at $\mathbf{u}$.*

Proof of Corollary 1. Since f is differentiable, all partial derivatives exist for all $\mathbf{v} \in [0,1]^n$, hence at $\mathbf{u}$. Hence, using previous theorem we can say the global minimum of f occurs at $\mathbf{u}$. □

Suppose the objective function is convex and all the partial derivatives exist at the obtained solution $\mathbf{u} \in [0,1]^n$ which is an interior point. Our algorithm terminates when two consecutive *runs* yield the same solution. It implies in the last *run*, the objective function values at all the sites obtained by making jumps of sizes $\delta_k = \frac{s_{initial}}{\rho^k}$ (until δ_k gets smaller than *step size threshold*) around $\mathbf{u}$, i.e. $f(\mathbf{u}_k^{(i+)})$ and $f(\mathbf{u}_k^{(i-)})$, are greater than or equal to $f(\mathbf{u})$ for $i = 1, \ldots, n$. So, taking *step size threshold* sufficiently small, our algorithm will reach the global minimum under the assumed regularity conditions of the objective function. From Corollary 1 we can conclude that if the objective function is convex and differentiable, taking *step size threshold* sufficiently small, our algorithm reaches the global minimum. It is to be noted that if the function is convex and differentiable and it takes minimum value at some interior point, evaluation of only the

first *run* is sufficient to obtain the solution.

Chapter 3

Simulation study and conclusion

For any global optimization algorithm, it is not always possible to reach the global solution while optimizing any non-convex function. So, the best way to compare the performances of several optimization algorithms is to perform a comparative study based on minimizing some well known benchmark functions.

3.1 Comparative study on Benchmark Functions

In this section, we compare the performance of our algorithm of global maximization on bounded intervals to five standard optimization methods: the 'Interior-point (IP)' algorithm, 'Sequential Suadratic Programming (SQP)' algorithm, 'Trust-region-reflective (TRF)' algorithm, 'Simulated annealing (SA)' ([14]), and 'genetic algorithm (GA)' ([1]). All these algorithms are available in Matlab R2014a (The Mathworks) via the Optimization Toolbox functions *fmincon* (for IP and SQP), *lsqnonlin*, *simulannealbnd*, and *ga* respectively. Out of these 4 available functions for optimization on bounded intervals, *fmincon* and *lsqnonlin* searches for local minimum and these are less time consuming in general. On the other hand *simulannealbnd*, and *ga* tries to find global minimum, being more time consuming. In our comparative study, we set the maximum number of allowed iterations and evaluations of objective function to be infinity for *fmincon*, *lsqnonlin* and *simulannealbnd* functions. For *ga*, we used the default values. Our proposed algorithm (GCDVSBI) is implemented in Matlab 2014a and the values of the tuning parameters have been taken to be default (as mentioned in Section 2.1). We did not consider the Nelder-Mead nonlinear simplex algorithm ([16]) and Levenberg-Marquardt gradient descent ([15]) in this comparative study as the implementations of these two methods in Matlab 2014a (via *fminsearch* and *fmincon* functions respectively) do not work for bounded parameter space. In the following sections, for each test function, the convergence is considered to be successful for a method if the absolute difference between the function value at solution returned by that

Table 3.1: Comparison of required time (in seconds) for solving Rosenbrock problem on bounded interval starting from $(x_1, x_2) = (-1.2, 1)$ using different methods.

Algorithms	Success	Time
GCDVSBI	Yes	0.175
SQP	Yes	0.131
IP	Yes	0.390
GA	No	0.515
SA	No	0.332
TRF	No	1.857

method with the true optimum function value is less than 10^{-2}.

3.1.1 Example 1 : Rosenbrock function on bounded interval

For Rosenbrock parabolic valley (2-D) problem, our objective function is

$$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2.$$

It attains the global minimum at $(x_1, x_2) = (1, 1)$. We considered $x_1, x_2 \in [-3, 3]$. The starting point is taken to be $(x_1, x_2) = (-1.2, 1)$. In Table 3.1 it is noted that only GCDVSBI, SQP an IP algorithm reach the global minimum.

3.1.2 Example 2 : Ackley's function

Ackley's function is given by

$$f(x, y) = -20 \exp\left(-0.2\sqrt{0.5\left(x^2 + y^2\right)}\right) - \exp\left(0.5\left(\cos\left(2\pi x\right) + \cos\left(2\pi y\right)\right)\right) + e + 20$$

where $-5 \leq x, y \leq 5$. The global minimum occurs at $(x, y) = (0, 0)$ For the comparative study of the proposed method with the other existing methods, each algorithm is started from 100 randomly generated initial points in the domain. In table (3.2), the average computation times and success rates have been provided for all methods. It is to be noted that only GCDVSBI, GA and SA algorithms reach the global minimum every time, but using GCDVSBI there is approximately 16 and 37 fold improvement in computation time compared to GA and SA algorithms respectively.

Table 3.2: Comparison of average computation time (in seconds) and success rate of reaching the global minimum for Ackley's function, Levi's function and Schaffer Function (N2 & N4) starting from 100 randomly generated starting points within the corresponding domains using different methods, for each functions.

Algorithms	Ackley's function		Levi's function		Schaffer function (N2)		Schaffer function (N4)	
	Success rate (%)	Avg. time	Success rate (%)	Avg. time	Success rate (%)	Avg. time	Success rate (%)	Avg. time
GCDVSBI	100	0.013	100	0.011	100	0.021	100	0.246
SQP	18	0.016	3	0.016	1	0.020	3	0.171
IP	21	0.038	8	0.034	1	0.052	3	0.349
GA	100	0.207	66	0.172	16	0.327	20	0.164
SA	100	0.477	83	0.444	6	0.428	12	0.378
TRF	22	0.074	1	0.557	NA	NA	0	0.187

3.1.3 Example 3 : Levi's function

Levi's function is given by

$$f(x,y) = \sin^2(3\pi x) + (x-1)^2\left(1+\sin^2(3\pi y)\right) + (y-1)^2\left(1+\sin^2(2\pi y)\right)$$

where $-10 \leq x, y \leq 10$. The global minimum is at $(x,y) = (1,1)$. Starting from 100 randomly generated points in the domain of (x,y), a comparative study similar to the above example is performed for this function also and the results of the comparative study have been provided in table (3.2). Again, it is to be noted that GCDVSBI outperforms other methods quite significantly, having the highest success rate with the lowest required computation time.

3.1.4 Example 4 : Schaffer function (N2 and N4)

Schaffer function (N2) is given by

$$f(x,y) = 0.5 + \frac{\sin^2\left(x^2-y^2\right) - 0.5}{\left(1+0.001\left(x^2+y^2\right)\right)^2}$$

and Schaffer function (N4) is given by

$$f(x,y) = 0.5 + \frac{\cos^2\left(\sin\left(\left|x^2-y^2\right|\right)\right) - 0.5}{\left(1+0.001\left(x^2+y^2\right)\right)^2}$$

where $-100 \leq x, y \leq 100$. The global minimum for these two functions are at $(x,y) = (0, 1.25313)$ and $(x,y) = (0,0)$ respectively. For these two functions also, a comparative study between various methods similar to the above example is performed and the results of the com-

Table 3.3: Comparison of required time and successful convergence for solving bounded Powells quartic function for various dimensions $N = 4, 8, 20, 40$ and 100, using various methods.

Algorithms	N=4		N=8		N=20		N=40		N=100	
	Success	Time	Success	Time	Success	Time	Success	Time	Success	Time
GCDVSBI	Yes	0.136	Yes	0.332	Yes	1.363	Yes	5.163	Yes	32.234
SQP	Yes	0.189	Yes	0.139	Yes	0.327	Yes	0.521	Yes	2.512
IP	Yes	0.387	Yes	0.478	Yes	0.557	Yes	1.562	Yes	2.663
GA	Yes	0.960	Yes	5.810	Yes	20.561	No	47.173	No	207.886
SA	Yes	0.896	No	2.735	No	8.253	No	38.760	No	134.133
TRF	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

parative study have been provided in table (3.2).Again for Schaffer function (N2) and (N4), GCDVSBI performs significantly better over all other algorithms. For optimizing Schaffer function (N2), using TRF algorithm results in unusually high (> 20 minutes) computation time for some cases. So the performance of TRF algorithm is not reported in case of Schaffer function (N2).

3.1.5 Example 5 : Bounded Powell's quartic function

Suppose our parameter is $\mathbf{X} = [\mathbf{x_1}^T,\ \mathbf{x_2}^T,\ \mathbf{x_3}^T,\ \mathbf{x_4}^T]^T$ is a vector of length $N = 4n$ and $\mathbf{x_i} = (x_{i1}, \ldots, x_{in})^T$ is a vector or length n for $i = 1, 2, 3, 4$. Then N-dimensional (N is a multiple of 4) Powells quartic function is given by

$$f(\mathbf{X}) = \sum_{i=1}^{n} \Big[(x_{1i} + 10x_{2i})^2 + 5(x_{3i} - x_{4i})^2 + (x_{2i} - 2x_{3i})^4 + 10(x_{1i} - x_{4i})^4 \Big]$$

The minimum value occurs at $\mathbf{X} = (0, \ldots, 0)^T$. We considered each element of $\mathbf{X}$ lies in $[-10, 10]$. The starting point is taken to be $\mathbf{x_1} = (3, \ldots, 3)^T$, $\mathbf{x_2} = (-1, \ldots, -1)^T$, $\mathbf{x_1} = (0, \ldots, 0)^T$ and $\mathbf{x_4} = (1, \ldots, 1)^T$. For $N = 4, 8, 20, 40$ and 100, this function is optimized by different algorithms setting the above-mentioned starting points (of corresponding dimensions). In table (3.3), the success rate of convergence and average computation time of different methods have been noted down. *lsqnonlin* returned some error in this case. Due to convexity of this function (see [19]), SQP and IP have extra advantage over other methods. It is noted that that, with increasing dimension, SQP and IP solve the problem most efficiently as expected. Among the global optimization methods, in higher dimensions, only our method reaches the true solution in reasonable amount of time outperforming other two global optimizers GA and SA.

3.2 Conclusion

This paper presents an efficient derivative-free algorithm for global optimization of functions of parameters coming from possibly distinct bounded intervals. The algorithm being derivative free, can be used efficiently for the functions whose derivatives are computationally expensive to calculate. Unlike other existing global optimization algorithms like GA and SA, the number of required function evaluations for this proposed algorithm increases only in the order of the number of parameters. Thus it works faster for high-dimensional global optimization problems. Another unique feature of GCDVSBI is the way it evaluates the objective function values at different sites, parallel computing can be incorporated easily and the requirement of parallelization only increases in the order of the number of variables unlike Monte Carlo methods where the requirement of parallelization increases exponentially with increasing dimension of parameter space (as mentioned in [13]).

For several benchmark global optimization problems, it is noted that GCDVSBI works 5-20 times faster on average than global optimization algorithms GA and SA and reaches the global solution more often. In Example 3.1.5 of Section 3.1, it is noted that for high-dimensional convex optimization problem, GCDVSBI reaches the global minimum in reasonable amount of time outperforming GA and SA. In Section 2.3 it has been shown that GCDVSBI reaches the global minimum in case the objective function is convex and differentiable and the global minimum occurs at any arbitrary interior point of the domain. Unlike other global optimization algorithms (e.g., GA and SA), GCDVSBI also efficiently optimize convex functions.

REFERENCES

[1] A. D. Bethke. Genetic algorithms as function optimizers. 1980.

[2] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, pages 1–52, 1996.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2006.

[4] R. H. Byrd, J. C. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

[5] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[6] T. F. Coleman and Y. Li. An interior, trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6:418–445, 1996.

[7] A. S. Fraser. Simulation of genetic systems by automatic digital computers i. introduction. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[8] L. Geris. *Computational Modeling in Tissue Engineering*. Springer, 2012.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Operations Research Series. Addison-Wesley Publishing Company, 1989.

[10] J. Goodner, G. A. Tsianos, Y. Li, and G.E. Loeb. Biosearch: A physiologically plausible learning model for the sensorimotor system, 2012.

[11] V. Granville, M. Krivanek, and J. P. Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, 1994.

[12] N. Karmakar. New polynomial-time algorithm for linear programming. *COMBINATOR1CA*, 4:373–395, 1984.

[13] C. C. Kerr, T. G. Smolinski, S. Dura-Bernal, and D. P. Wilson. Optimization by bayesian adaptive locally linear stochastic descent. http://thekerrlab.com/ballsd/ballsd.pdf.

[14] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *Australian Journal of Biological Sciences*, 220:671–680, 1983.

[15] D. W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11:431–441, 1963.

[16] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[17] J. Nocedal and S. J. Wright. *Numerical Optimization*. Operations Research Series. Springer, 2nd edition, 2006.

[18] F. A. Potra and S. J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 4:281–302, 2000.

[19] T. Steihaug and S. Suleiman. Global convergence and the powell singular function. *Journal of Global Optimization*, 56(3):845–853, 2013.

[20] M. H. Wright. The interior-point revolution in optimization: History, recent developments, and lasting consequences. *Bulletin of American Mathematical Society*, 42:39–56, 2005.