Proposing Effective Coordinate Search Methods for Solving Large-scale Expensive Black-box Optimization Problems

by

Ehsan Rokhsatyazdi

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in **Electrical and Computer Engineering**

Department of Electrical, Computer, and Software Engineering (ECSE)

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

August 2020

Thesis Examination Information

Submitted by: **Ehsan Rokhsatyazdi**

Master of Applied Science in **Electrical and Computer Engineering**

Thesis title: Proposing Effective Coordinate Search Methods for Solving Black-box Large-scale Expensive Optimization Problems

An oral defense of this thesis took place on August 20, 2020 in front of the following examining committee:

Examining Committee:

Chair of Defense Committee	Dr. Ying Wang
Research Supervisor	Dr. Shahryar Rahnamayan
Examining Committee Member	Dr. Jing Ren
External Examiner	Dr. Ken Pu

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies

Abstract

In engineering and science, optimization plays a vital role in many real-world applications. In this work, several novel optimization algorithms based on Coordinate Search (CS) algorithm are proposed. CS is a gradient-free technique and we have enhanced them for solving Black-box, non-convex, and expensive large-scale problems. These CS-based algorithms can handle mixed-type variables. When an optimization problem is large-scale and expensive, it is a very challenging problem to solve because it is intersecting two conflicting properties. Large-scale problems require extensive fitness evaluations, but each evaluation is time consuming. It gets more challenging when the budget is limited, which is the case in most real-word applications. The proposed CS-based algorithms reduce the search space exponentially; this makes it a powerful method for optimizing high-dimensional problems with limited budget. The proposed algorithms show a very promising performance on optimizing high-dimensional problems; tested on the CEC-2013 benchmarks problems and neural network training.

Keywords: coordinate-search; gradient-free; non-convex; neural-network; large-scale optimization; expensive optimization

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

21/08/2020

Ehsan Rokhsatyazdi

Statement of Contributions

For non-convex, black-box, large-scale problems, there is no effective optimization algorithm. In this work, we proposed algorithms based on CS for optimizing such problems. Results show superior performance on the CEC-2013 benchmark in front of the state-of-the-art algorithms. Moreover, it can be an alternative method for training neural networks with a large number of weights on a machine with a single CPU, which was just feasible by gradient-based algorithms.

Sole Authorship

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

Acknowledgments

I would like to express my heartfelt thanks to my supervisor Prof. Shahryar Rahnamayan and my co-supervisor Prof. Hamid Tizhoosh, for their honest advice and support Throughout my study at Ontario Tech university. They have both been knowledgeable, charismatic, and amazing teachers. They shed light on my educational way in this journey. Also, I learned lots of valuable life lessons, such as being humble, patient, tolerant, and being open-minded. I would not be where I am today without their support. I always appreciate their constructive feedbacks and criticism. They were very patient with me and helped me step by step to overcome all problems during this journey.

Finally, I would like to thank my parents and my sister for supporting me throughout my education and making many sacrifices to ensure that I could pursue higher learning and achieve my dreams. I would like to dedicate this thesis to them, who meant the world to me, and always patiently supported me and guided me during tough days.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

AAT	All-At-Time
AC	Ant Colony
BCD	Block Coordinate Descent
BSF	Box-constraint Shrinkage Factor
CGD	Coordinate Gradient Descent
CR	Crossover Probability
CS	Coordinate Search
DE	differential Evolution
EA	Expansion amount
ES	Expansion Sequences
FC	Fitness Call
FE	Function Evaluation
GA	Genetic Algorithm
GD	Gradient Descent
GSS	Generative Set Search
LHS	Latent Hypercube Sampling
LS	Left Shift
LSGO	Large Scale Global Optimization
MCS	Multilevel Coordinate Search
NFC	Number of Fitness Call
NFE	Number of Function Evaluation
OAT	One-At-Time
PSO	Particle Swarm Optimization
RCD	Randomized Coordinate Descent
RPCD	Random Permutation Coordinate Descent
RS	Right Shift

SCF	Stochastic Coordinate Frequency
SCS	Stochastic Coordinate Search
SCSF	Stochastic Coordinate Search Frequency
SGD	Stochastic Gradient Descent
SPB	Selected Portion of Box-constraint

Chapter 1

Introduction

1.1 Introduction to Optimization

In most applications, processes, and research works ranking from engineering to medical applications, there are some parameters, which need to be adjusted to achieve a better performance, accuracy, and/or efficiency. All these efforts are formalized as an optimization problem, making optimization almost everywhere, and in every application. "Optimum" is a Latin word which means "the ultimate ideal," and "Optimus" means "the best." So, optimization means to move whatever we are coping with towards its best state [1]. According to this definition, enhancement and optimization are two sides of a coin.

The main part of this work is based on developing an optimization method and investigating its applications for large-scale non-convex problems. Therefore, we start with reviewing common concepts in the optimization field, which is used in different parts of this work. In the coming sections and subsections, the main concepts and definitions will be described.

The process of making a design, decision, or system as fully perfect, effective, or functional as possible is called optimization. An optimization problem describes a problem in which an objective can be minimized or maximized. The problem should be modeled with one or more defined fitness (objective) functions to measure the performance of the optimization process. As such, modeling a problem and determining a fitness function are crucial tasks because the performance and accuracy of the results correlate with the accuracy of the modeling. If it is possible, we try to have an exact model for problems, but in some cases, finding the exact model is impossible or complicates the problem because many parameters are involved. In these situations, the approximate model is preferred, but the approximate models do not represent the problem exactly, and in some situations, the model might fail. Therefore, when using the approximate model, we need to be more cautious about the results of optimization. The formal definition of a single-objective optimization method is shown in Eq. (1.1), and Ep. (1.2). $f(\chi)$ represents the objective function, χ stands for variables, $g_i(\chi) \leq 0$ means inequality constraint, and $h_i(\chi) = 0$ is called the equality constraint.

$$\text{Minimize/maximize: } f(\chi) \tag{1.1}$$

$$\begin{aligned} \text{Subjected to: } & g_i(\chi) \leq 0 \quad i = 1, \dots, m \\ & h_i(\chi) = 0 \quad i = 1, \dots, p \end{aligned} \tag{1.2}$$

The optimizer tries to minimize (or maximize) the fitness value, which can be obtained from the fitness (objective) function. Some problems only have one objective function (or some correlated objectives that can be simplified to one objective) such as, minimizing the cost of a product. In contrast, multi-objectives optimization problems have more than one objective which creates conflict. For instance, buying a house with a lower price and living in a higher area are two conflicting objectives, because lower-priced houses lead to lower areas. In this work, we optimize single-objective problems.

Variables (or inputs) that contribute to the fitness function define the dimension of the problem. For example, if the number of inputs of our function, which is required to minimize, is seven, the dimension of the problem is also seven. Variables may also come in different types, such as binary, integer, real, and categorical. In some problems, different types of variables cause mixed-type optimization problems. In the most cases, increasing the number of variables increases the complexity of the problem exponentially. Thus, if possible, we try to decrease the dimension of the problem using different methods such as, feature selection and/or bundling variables.

Each variable can variate under some constraints. For example, if a tile is designed with the highest area value, the manufacturer can limit the length and width of the tile within a feasible range. These limitations on variables are known as box-constraint. Other constraints also exist such as, limitations on one variable based on another variable amount. For example, the summation of two variables should not be more than a specific amount. All these types of constraints should be considered during optimization.

Based on a fitness function, variables can make a specific space called a search space, with the dimension equal to the number of variables. If the problem has just two variables, the search space will be a surface in a three-dimensional map (the third dimension is fitness value), and the optimizer tries to find the minimum or maximum point on this surface, called landscape.

Some solutions can be found in search space, which are globally the best optimum points in terms of their objective values. These points are the global minimum or maximum. Some solutions in their local space around themselves are optimum. These points are considered as the local minimum or maximum.

In some search spaces, the local and global optimum is the same, with only one optimum point. These are considered unimodal and convex [2],[3] problems. If a search space has some local and global optimums, it is considered as multimodal and non-convex [4] search space. In non-convex search spaces, the optimizer is likely to trap in a local optimum instead of finding the global solution. None of the algorithms can guaranty that it will find the global solution, but some algorithms have this ability to find the global one under some circumstances, which are heavily related to the landscape shape and the type of problem.

Variables that contribute to the fitness function can be separable or non-separable. In a fitness function with separable variables, the fitness function can be minimized or maximized with respect to one variable, while keeping the other variables fixed, this process can be done for each variable one-by-one. But in non-separable fitness functions, it cannot be minimized or maximized by changing variables one by one. Therefore, non-separable fitness functions are harder problems to be optimized.

In some problems, the fitness function of the optimization can be calculated or can be estimated. But in some cases, which are called black-box optimization, the fitness function as a mathematical model is not available, and we only have access to the inputs and outputs of the function or system. Therefore, by changing the inputs and observing the outputs, the optimizer changes the parameter to find the optimum solution. If we have access to an identical optimizer, we will find the optimum solution for problems with either a mathematical fitness function or a black-box optimization. In this work, we try to develop an optimization algorithm to tackle the black-box problems.

Based on mathematical proof, the derivative (or gradient for more than one dimension) of a fitness function can find the optimum points of that function. This method is very useful if the fitness function is differentiable; but for non-differentiable functions, it cannot be used. In many problems, we struggle with discrete or categorical variables as well as mix-type variables. None of these problems can be addressed by gradient-based optimization algorithms. Since this is a decisive factor for the optimization process, we can categorize optimization algorithms into two main categories; gradient-based and non-gradient-based (gradient-free) algorithms. We will go through them in the next sub-sections.

1.2 Gradient-Based Optimization

Gradient-based optimization works based on derivative (for one-dimensional functions), and gradient (for more than one-dimensional problems) [5]. There are two approaches to use the gradient for optimization. The first one is applicable when the fitness function is available in the mathematical format, and the derivatives and gradients can be mathematically calculated. In these cases, the first and second derivatives are obtained. So, the optimum solution can be found mathematically. When the first derivative (or gradient) of a function is equal to zero at a particular point, it means that the point is the maximum or the minimum or a stationary point of the function. At this point, if the second derivative is greater than zero, this point is an absolute minimum. If the second derivative is lower than zero at this point, this point is an absolute maximum. However, if the second derivative is equal to zero, this point is a stationary point. The second approach is used when the function cannot be calculated in a mathematical format, or it is very complicated to find the formula for the fitness function. In these conditions, numerical calculation helps find the solution [6].

A useful and important techniques based on the gradient is called the Gradient Descent (GD). The GD finds the gradient of the initial point (mostly uniform random initial point), and then moves toward (backward) the gradient vector to find the values of variables, causing the maximum (minimum) value of the fitness function. The rate of this movement towards the optimum point—defined as the learning rate— needs to be selected at the right amount. GD can find the optimum point very fast, but a disadvantage of the GD is how it traps in the local optimum of a non-convex function. Moreover, the GD cannot be used for discrete and non-differentiable functions.

Coordinate gradient descent (CGD) can be considered as a variation of the GD, which just moves based on a (or a subset of) variable derivative (gradient). Details of gradient-based techniques are explained in the literature review section.

1.3 Gradient-Free Optimization

If there is no access to the gradient, or there is a desire to avoid trapping in the local optimum, Gradient-free algorithms can play a vital role. We can categorize the most important algorithms in this regard into these main categories:

The first one is the metaheuristic algorithms. In metaheuristic approaches, one or more acceptable solutions can be obtained, but they do not guaranty finding the optimum solution. One of the most famous metaheuristic algorithms is a genetic algorithm (GA), which is proposed by John H. Holland in 1976 [7]. The GA can be considered as a parent of a large portion of metaheuristic algorithms. One of the very successful variations of GA is differential evolution (DE) [8], which use fundamental operation (crossover, mutation) like GA, but there is some modification in these operations and selection.

Another algorithm is the Coordinate Search (CS) algorithm [9],[10],[11],[12],[13],[14]. One variation of CS is also known as compass search, which belongs to a generating set search (GSS) algorithms. This method is based on changing one or a group of variables in each step towards the optimum solution, and keeps other variables fixed in a cyclic manner. CS has some remarkable advantages. It has the ability to find the global solution (not trapping in the local optimum) and needs only a few numbers of FE in comparison with GA and DE. So, CS can be a good candidate for high dimensional optimization compared to other metaheuristic methods which demands large number of Function Evaluation (FE) demand. Moreover, CS can be used for non-differentiable, discrete, and non-smooth functions. Because of these abilities and advantages, in this work, we developed the CS algorithm for large-scale global optimization (LSGO). Metaheuristic algorithms are explained in detail in the literature review section.

Another gradient-free approach is swarm algorithms, which are based on the colony of the spices like ant and bee, and inspired from the behaviors of a flock of birds or fish. There is some kind of intelligence in the swarm of spices, which leads to a better ability to find food as well as better protection. Finding food is considered as an optimization process; researchers realized that they could use this ability for optimization purposes. Two of the most famous algorithms in this field are particle swarm optimization (PSO) [15] and the ant colony (AC) [16]. In our research, because we do not work in this area, swarm algorithms are not explained in detail, but they are used in some works to compare the results.

1.4 Main Limitations in the Optimization Process

As mentioned above, the optimization process has some limitations which need to be very carefully considered for each specific optimization problem. In this section, we mention some of the most important limitations, and based on them, we clarify the motivations for this work; limitations and problems always open the door to researchers to improve the process and techniques.

Most important limitations and challenges in the optimization process are summarized in the following categories:

a- Budget limitations

For evaluating each optimization step, some Number of Function Evaluation (NFE) or Fitness Call (NFC) is needed. In many cases, it is expensive in terms of time, computational power, energy, material, or physical demand. So, in the optimization process, we try to keep the NFE low. But with limited NFE, finding the optimum solution becomes challenging. Therefore, researchers try to develop algorithms, which need less NFE to find the solution or attempt to find a better solution by a low NFE. Performance plots show which algorithm finds the solution faster, and which algorithm finds better solutions.

b- No-Free-Lunch-theorem (NFL)

The shape of the search space is another confinement factor. For different landscapes, various algorithms perform better. Based on the no-free-lunch-theorem [17], there is no algorithm that performs best overall optimization problems. For each problem, there are some special techniques that perform better. But, researchers try to develop approaches that perform better on a wide range of problems. Therefore, for testing each algorithm, performance on a group of functions (not just one function) is measured to develop better algorithms that can solve more problems and be more generalized. But we should notice that one algorithm might work very well for specific problems. For example, the gradient-based-algorithm works very well for convex and smooth landscapes. However, for non-convex and multimodal landscapes, it is more likely to be trapped in a local optimum. So, if we discover the best ability of a method, we can use the method in applications and problems which are matched with the method.

c- Robustness of performance

Metaheuristic and stochastic gradient descent algorithms are not deterministic, meaning that when each time the algorithm is run, a different result is achieved. If the standard deviation between the results is very high, it shows the algorithm does not behave robustly. In many applications such as industrial applications, robustness is very important. So, if the stochastic algorithms find the solution with a lower standard deviation for different runs, the solution is more desirable. Furthermore, we report standard deviation of each function and method for benchmarks. Also, robustness is measured based on the variety of problems which the algorithm is able to solve.

d- Decision making is based on previous steps in the optimization process

In many cases, the historical data during the optimization process is used to make decisions for the next steps. But, from the optimizer point of view, landscape changes in each step, because, the optimizer explores a different part of the landscape in each step, making it a limitation. If an algorithm can make more generalized decisions based on previous steps, it is more likely to find better results. If we know the shape of the landscape, we can make more effective decisions. But for black-box optimization, there is no knowledge (or very limited knowledge) about the search space and more generalized decisions bring about superior performance.

e- The complexity of the optimizer

We try to develop an optimizer, which is not very complex because a very complex optimizer is less likely to be used for different applications. One of the advantages of the GA algorithm that makes it very popular was the simplicity of it. A very complex algorithm cannot be well understood, so developing and leveraging it can be problematic.

1.5 Motivation and objectives

There is a lack of optimization techniques for non-convex black-box large-scale problems. Our motivation was developing a more generalized optimization algorithm which can optimize differentiable and non-differentiable, separable and non-separable, convex and non-convex, unimodal and multimodal, smooth and non-smooth, discrete and continuous, and grey-box and Blackbox high-dimensional problems. As explained above, we were looking for an uncomplex, more generalized, and gradient-free algorithm. So, after many experiments and research, we found a less developed but very potent algorithm. This method is a coordinate search (CS) optimization technique that has not been developed properly. In this work, we developed CS methods and proposed several schemes for low budget and high budget optimization applications. Moreover, we compare our results with the state-of-the-art algorithms from 2013 to 2019 on CEC-2013 benchmarks, and it shows the remarkable ability of our proposed algorithm. As a very high dimensional optimization case study, fully connected neural networks were trained by our proposed CS algorithm. As Table 1.1 illustrates, if we consider four quarters in the optimization area, GD can be used for low and high dimensional problems which are convex and differentiable. The Differential Evolution (DE) algorithms can be used for low dimensional non-convex problems, but for high dimensional (large-scale) and non-convex problems there are no such effective methods except the proposed method. We also target black-box optimization, so the algorithm can be used in many different applications. The downside is that for black-box optimization, there is no mathematical proof for convergence. So, we attempt to empirically show the advantages of the proposed method.

Table 1.1: Target zone for the proposed method in optimization

	Low-dimensional	**High-dimensional**
Convex differentiable	GD CD	SGD
Non-convex Differentiable / Non-differentiable	DE	Proposed CS

1.6 Thesis Outline and organization

In Chapter 2, previous works will be reviewed, and the advantages and disadvantages of existing methods will be discussed. Then in Chapter 3, we try to develop a gradient-free algorithm for non-convex large-scale black-box optimization. We start with a basic model and then enhance it to achieve an efficient algorithm. In order to evaluate the method, in Chapter 4, the best-proposed scheme in Chapter 3 is tested on the CEC-2013 benchmark to compare the results with four state-of-the-art algorithms. Chapter 5 is about the application of the proposed method in neural network training. Conclusion and future works are reported in Chapter 6.

Chapter 2

Literature Review

In this chapter, more details of the well-known algorithms either which are related to the proposed method or can be mentioned as the best competitor, are explained. Moreover, previous works on coordinate search algorithms, which is the base of our method, are investigated. The pros and cons of each one is also mentioned.

2.1 Gradient-Based Algorithm

Optimization algorithms, which work based on the gradient, are very well-known and effective. We can use some of the enhancement techniques in gradient-based algorithms to improve CS ones. In this sub-section, different variations of gradient-based methods are explained.

2.1.1 Gradient Descent (GD)

Gradient descent is one of the most famous optimization algorithms, especially for neural networks. This method is a gradient-based method, so the function which should be optimized is required to be differentiable and smooth. Moreover, stationary points and areas in search space can affect the optimization process and convergence speed, leading to premature convergence.

Gradient shows the direction towards the optimum point, but for descending or ascending to it, step size also plays a vital role. If the step size is selected too small, reaching the optimum point will need a lot of steps, and the convergence will be very slow. On the contrary, if the chosen step size is large, when it comes close to the minimum point, it will jump around it and will not reach the solution or even jump to another part of the landscape, losing the solution. So, by choosing a large step size, it is impossible for GD to reach the solution [18]. One way to have a reasonable speed to reach the target, and also not miss it is to benefit from adaptive step size. To have an adaptive step size [19], GD algorithm uses both gradient and learning rate to adjust the step size towards the target point. The learning rate should be adjusted by the user or other algorithms, and it brings other concerns to the scene. Choosing an appropriate learning rate can be hard because it is a problem-dependent parameter. If the problem changes, the learning rate should be changed. Moreover, different parts of the landscape may have totally varied shapes, so it needs a different learning rate even for a single problem.

For high-dimensional optimization problems, calculating gradient is costly in terms of computational cost. For each step toward the target, it needs to calculate derivatives with respect to each variable, and for high-dimension problems, it would be very demanding in terms of calculations. Therefore, in high dimension problems such as neural network optimization, instead of calculating derivatives with respect to all variables, some of the variables are randomly selected, and then the gradient is calculated with respect to them, not all variables. This method is known as stochastic gradient descent (SGD) [20],[21],[22], and recently because of the widespread usage in neural networks, it has gotten much attention. This method actually approximates the model, but can severely reduce the calculation burden, and make the process very fast. But each method has its own flow, and in this case, randomly selecting some variables makes the algorithm not-deterministic, creating a different solution every time we run it. In some cases, it is a really challenging issue. In neural network hyperparameter optimization, whole network should be trained to find the performance of the network during each step. But because of the stochastic nature of the training process by SGD, for each evaluation, the network should be trained a couple of times (e.g., five to 100 times), and then getting the average to find the actual performance of the network. It makes the process slower and more computationally intensive.

SGD or stochastic approximation has many different variations [18]. The classical SGD uses each sample in the dataset to update the solution. In Eq. (2.1), α_k represents the learning rate, and $\nabla f(X_k, 1)$ represent the gradient of the function in X_k by considering one data sample.

$$X_{k+1} = X_k - \alpha_k \nabla f(X_k, 1) \tag{2.1}$$

Mini-batch SGD is another version of SGD, which is using an average of gradient over a specific number of samples (batch size), to update the solution. In Eq. (2.2), $\nabla f(X_k, m)$ stands for an average gradient of the function in X_k by considering the average gradient on m data samples. By selecting m based on the optimization stage, an adaptive and dynamic version of mini-batch SGD can be adopted [18].

$$X_{k+1} = X_k - \alpha_k \nabla f(X_k, m) \tag{2.2}$$

Last but not least, the GD does not apply to categorical and discrete functions for optimization purposes. These kinds of functions are not differentiable, so the GD cannot be used in these situations. For black-box optimization with GD, numerical approaches can help estimate the gradient at each point.

The most important problem of the gradient-based methods is trapping in local optimums. There are some techniques to prevent tapping in the local minimum. One of the most conventional action is random initialization. Random initialization helps the algorithm start from different points, and in some cases, these points are in the region of interest in the search space. In other words, these points are in a unimodal part of the search space, which contains the global optimum. So, the gradient descent algorithm can gradually move toward the minimum point in this unimodal environment.

2.1.2 Coordinate Descent (CD) or Coordinate Gradient Descent (CGD)

CD and CGD are iterative techniques, which minimizes the fitness function with respect to one or a group of variables in each iteration. Other variables are kept fixed in the current iteration. This method breaks the problem into some lower-dimensional subproblems which can be solved more easily than the main problem [23], Tseng [24], Luo, and Tseng [25], [26], and Bertsekas and Tsit Siklis [27] came up with remarkable contributions to the convergence characteristics of coordinate descent in the 1980s and 1990s.

CGD [28],[29],[30], is considered a subcategory of GD. This method is gradient-based, but instead of moving by the gradient vector to proceed to the optimum point, CGD in each step moves with respect to one variable while keeping other variables fixed. One-by-one, CGD changes all variables in a cyclic manner to reach the solution. CGD and GD for unimodal search space lead to the same result, but for non-convex search space, they may end up to different solutions. Therefore, in some problems, GD works better, and in other ones, CGD performs better. For example, in linear regression and LASSO regression, CGD performs better than GD [31]. Eq. (2.3) shows how CGD updates each variable. α_k represents the learning rate, and $\nabla f(X_k)i_k$ represents partial gradient with index k of the fitness function in X_k, and $e_{i_k} = [0,0,\ldots,1,\ldots 0]^T$ to make a vector just with one none zero element. This none zero element is the variable which should be updated in the current iteration [32].

$$X_{k+1} = X_k - \alpha_k \nabla f(X_k)_{i_k} e_{i_k} \tag{2.3}$$

A version of the CD which attracts a lot of attention is called Randomized CD (RCD). In this version, each variable is chosen randomly to update in each iteration. The scheme can be thought of as sampling with replacement. This change helps the algorithm have better exploration [23], but in some cases, the CD outperforms the RCD [33], and it is completely problem-dependent. Another scheme is random permutation. If we consider the set of variables as a chromosome, a different order of the genes in the chromosome can be considered as different permutations, defined as Random Permutations Cyclic CD (RPCD). Some problems (like convex quadratic optimization problems) show better performance than RCD [32],[34],[35]. Random permutation helps the optimizer search different points in the landscape, and the chance of finding a global solution is higher than the CD. In general, we can say the performance of these schemes to find the solution is task-dependent, and we cannot select the best method. The best method for various problems can be different.

Another important subject is the ratio of convergence (ROC). The ROC shows how fast an algorithm can find the solution in comparison to the other ones. Experimental result in [32] shows that CD is the fastest one, RPCD is the second fast, and RCD is the slowest one. Intuitively, we can say CD finds the optimal (or local-optimal) solution in a fast way because from the optimizer's point of view, the landscape does not have a very harsh change. For RPCD, because the landscape in each iteration, changes for the optimizer, and the optimizer face a new search space, it takes more time to converge. For RCD some coordinates may rarely show up, So, convergence takes more time even more than RPCD.

In terms of calculation, CGD is less intensive in comparison with GD [36]. CGD only needs to calculate one derivative with respect to one variable in each step and keeps other variables fixed; but GD needs to calculate gradient with respect to all variables.

Speed of the GD in high-dimensional problems such as the optimal design of neural networks can be more than CGD because GD changes all variables at the same time (all-at-time (AAT)), while CGD just changes one-at-time (OAT). For example, in neural networks, for each epoch in GD, all training data should be fed to the network one time. However, for CGD training data should be fed to the network one time for each variable, and there are thousands or millions of variables. Thus, CGD is more demanding than GD, but by some tricks to bundle or group variables, and then CD can converge a couple of times faster.

When we are dealing with a high-dimensional problem, we can have a scholastic version of CGD, as we mentioned SGD before. In stochastic CGD (SCGD) [37], instead of considering all variables, a random subset of whole variables participates in the CGD process. These partially updating variables in CD are known as an accelerated version of CD [36]. Then in the next iteration, another subset will take part in the optimization procedure. This approach gives almost an equal chance to all variables to play their roles to reach the optimum solution while simultaneously reducing the complexity of the problem, and speed up the process. In another accelerated version, the subset of variables is selected based on the sensitivity of the fitness function to variables, which means variables with more sensitivity have a higher chance of being selected. Moreover, each variable can appear in the subset of essential variables more than one time. Meaning in a singular iteration. The optimizer can go further in a selected direction than other directions [38]. Moreover, the random selection of the subset of variables results in finding a different solution in non-convex search space, which is helpful to avoid tarping in local minimums, and bring about better exploration.

Another well-known version of CD is block coordinate descent (BCD) [39],[40],[41],[42]. In this version, variables are grouped in different blocks, and for optimization, one block changes when other blocks are fixed. BCD can be applied for problems in which variables are partitioned into different blocks, For example, non-negative matrix factorization [43], group LASSO [44], and some distributed computing problems, which blocks of variables appear naturally [45].

BCD can be used for large scale problems. It treats a group of variables in one iteration, but it needs to model the problem in a way where separated or loosely connected partitions appear. To achieve this model, a mathematical model of the problem should be available, and then by some approximation, convert the problem to a partially separable problem; then BCD can be used for optimization [46]. These conditions limit the applications of the BCD. For example, in black-box problems that have no mathematical model available, BCD approaches cannot be used. One solution in this situation can be using grouping algorithms to decompose variables [47]. So, the problems will be decomposed into some sub-problems, and then BCD can optimize them. But, this decomposition process needs some extra fitness calls, which are not desirable. Moreover, instead of optimizing the original problem, BCD optimizes the approximate model, which is not accurate in some special situations, and leads to some incorrect or infeasible solutions.

2.2 Gradient-Free Algorithm

Calculating the gradient for optimization causes some limitations and problems which are mentioned above. Moreover, in some situations, such as discrete and non-differentiable functions, the gradient cannot be calculated. For having a global optimization which can be applied for all kinds of functions, gradient-free algorithms can be used. In this section, we explain the characteristics of the important ones.

2.2.1 Metaheuristic Optimization

In metaheuristic approaches, one or more acceptable solutions can be obtained, but they do not guarantee finding the optimum solution. Therefore, metaheuristic approaches can be used for a wide variety of applications. The most popular metaheuristic algorithms are explained in this part.

The Genetic algorithm [7] is one of the most famous metaheuristic techniques, and other evolutionary algorithms that try to imitate evolution in species can be considered as variations of GA. There are many variants of the GA which use a variety of crossovers, mutations, adaptive population sizes, and more. In these works, mainly the way of using crossover or mutation is modified or enhanced, tailoring them for a specific problem. One of the powerful methods that are strongly connected to the GA is Differential Evolution (DE). We will take advantage of this method in some applications in this work, so it will be fully explained in the next section.

DE can be considered as one of the variations of the GA and performs way better than the classical version of the GA in many cases. It is efficient, simple, and easy to implement. DE benefits from mutation and crossover. It starts with generating a random population, just like GA. But in DE, three parents take part in the mutation. Then by using Eq. (2.4), the mutant vector is calculated. In Eq. (2.4), X_r represents one of the randomly selected solutions (chromosomes). $V_{i,G+1}$ stands for mutant vector, and F is the mutation factor, and shows how much the difference between the two other solutions can change the mutant vector. There is an interesting fact behind this mutation approach. In the exploration phase of the optimization, differences between X_{r2} and X_{r3} are high, so the mutant vector changes severely to have better exploration over search space, but when the optimizer is in the exploitation phase, differences between X_{r2} and X_{r3} are low. Therefore, the mutant vector will not change drastically, and it is better for fine tuning. As a result, we can say the mutation operator in DE is self-adaptive and can automatically distinguish the exploration and exploitation phase, and then can change its behavior based on it.

$$V_{i,G+1} = X_{r1} + F \times (X_{r2} - X_{r3}) \tag{2.4}$$

The second operator in DE is crossover. This operator can generate a new chromosome by selecting genes from a mutant vector. For selecting each gene, the crossover operator uses *CR,* which is the crossover probability of selecting a gene. If *CR* is smaller than a randomly generated value, a gene from a mutant vector is selected. But if the *CR* is higher than the randomly generated value, the gene is chosen from the parent vector. Eq. (2.5) shows how the crossover works. U_{G+1} stands for vector after crossover operation.

$$U_{j,i,G+1} = \begin{cases} V_{j,i,G+1}\ if \quad rand_{j,i} \leq CR \ \ or \ \ j = I_{rand} \\ X_{j,i,G+1}\ if \quad rand_{j,i} > CR \ \ or \ \ j \neq I_{rand} \end{cases} \tag{2.5}$$

The last phase in the DE algorithm is selection. In the selection stage, based on the fitness of the U_{G+1} and X_G one of them which has a better fitness value is selected. Eq. (2.6) can be used for selection in DE. $\mathfrak{f}(.)$ represent the function that should be optimized.

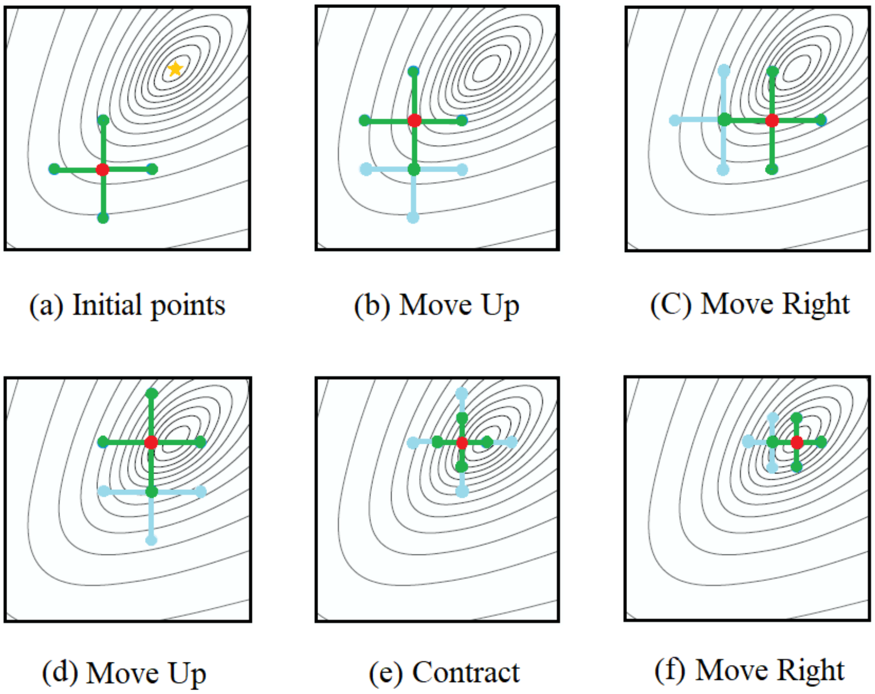
$$X_{i,G+1} = \begin{cases} U_{i,G+1} \ \ if \ \ \mathfrak{f}(U_{i,G+1}) < \mathfrak{f}(\,X_{i,G}) \\ X_{i,G} \ \ otherwise \end{cases} \tag{2.6}$$

2.2.2 Coordinate Search (CS)

In the CS method, one variable or one block of the variables changes during the optimization process, and other variables and blocks are kept fixed just the same as the CD. However, the CD is based on the gradient. In contrast, CS is a non-gradient based method and just works by sampling some points from each coordinate or block of variables.

One famous branch of CS algorithms is the compass search that belongs to generating set search (GSS) algorithms [10],[11],[12], [13]. In this type of algorithm, in each initial point, the algorithm looks forward and backward in each coordinate separately by using a defined step size. If the fitness value improves more than a threshold, this is considered a successful search, and the initial point moves toward the successful direction. Otherwise, the step size becomes smaller (like learning steps in the GD, which gets a smaller value to not jump around the optimum point), and this process repeats again until the step size becomes smaller than the predefined value, which is the stopping criteria of the algorithm. Figure 2.1 illustrates the different sequences in this method. The target point is shown by the yellow star. Red dot in part (a) shows the initial point. The algorithm takes some samples from four sides of the initial point, then moves toward the best one (b), and keeps searching until no desired progress is achieved. Then in contract phase (e), the radius of search points is shrunken. It keeps searching until no improvement is achieved which is the stopping criteria.

This algorithm is more likely to tarp in a local optimum because, in this technique, some points around the initial point are explored while other parts of the landscape will not be explored. We are interested in the algorithm which is less likely to be trapped in a local optimum. This method cannot be used as a base method for our work. Moreover, test results in Chapter 3 show that for large-scale global optimization, it does not perform well.



(a) Initial points

(b) Move Up

(C) Move Right

(d) Move Up

(e) Contract

(f) Move Right

Figure 2.1: Compass search procedure [11], the yellow star is the optimum point, and compass by moving and contraction moves toward it

The method proposed in [14] is a multilevel CS (MCS), which is a global search method. In them, there are three phases, initialization, splitting, and local search. In this method, for the first variable (or the axis), some points like the boundary of box-constraint, middle point, and some other points (which are introduced as golden points) are considered. Then we split the search space based on the fitness value. After splitting, a region with more variation in the fitness value is selected for the first variable, and then by considering the selected region for the first variable, this procedure is repeated for the second variable. For the second variable a suitable region is selected, and then repeats for other variables. So, the region of interest, after some iterations, shrinks to a very small region, and then using local search, tries to find the global optimum. Figure 2.2 shows these steps. Then the algorithm searches other parts of the landscape one-by-one to find another global optimum. This method has been tested by different functions, but in all cases, used for problems with less than 72 variables (to the best of our knowledge), which cannot be considered as high-dimensional problems.

Another disadvantage of this method for high (or huge) dimensional problems is the number of sampling points, which is five for each variable in each iteration. So, it needs $5 \times D$ fitness calls (2.5 times more than our proposed method). Moreover, in very large-scale problems, the landscape is huge in size, and there is no way to search all parts of it with limited sampling. Another problem that has been explained in the Chapter 3 emerges when a search algorithm selects a small portion of the search space by a relatively small number of sampling. This action increases the chance of losing the optimum global region. Thus, in our best version of the proposed method, we avoid this action and try to gradually reduce the size of the search space to find the region of interest. Results on benchmarks confirm that MCS for large-scale optimization cannot beat the proposed CS.

In [9], MCS results are compared with other algorithms like PSO and CMA-ES. Results show that PSO and CMA-ES perform better for low-dimension (4 to 72 variables for different problems), but the CS algorithm in early iterations shows more improvement. So, if it is low-budget in terms of NFC, CS can find a reasonable solution. However, if there is enough budget for NFC, other methods like PSO and CMA can find a better solution, which is the case for low dimensional problems. For high-dimensional problems, because the search space grows exponentially, metaheuristic algorithms cannot perform very well. Therefore, in problems like neural network training with hundreds of thousands of weights, the budget that metaheuristic algorithms need to find the solution is extremely high, which is not feasible. Therefore, they cannot be used for these kinds of problems, but proposed CS can find a reasonable solution with low-budget, making CS a good candidate for very high dimensional problem optimization.

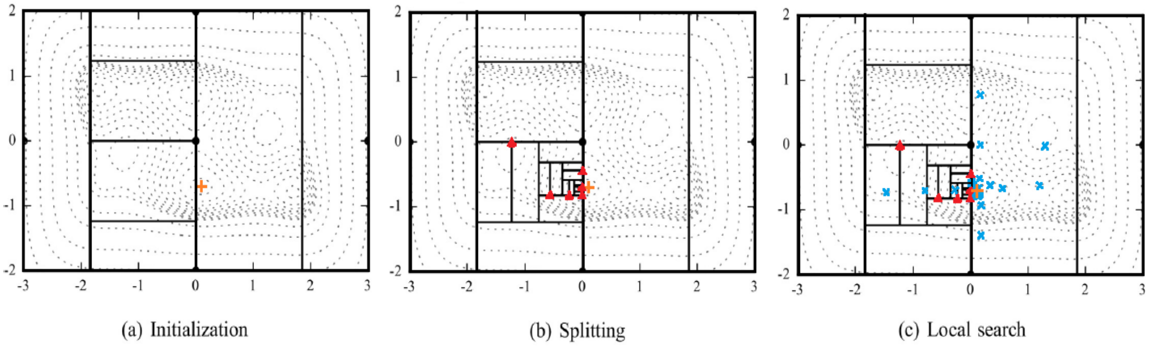


Figure 2.2: Multilevel coordinate search procedure [9]

In some research work, they tried to combine CS with another algorithm to improve the performance of search algorithm such as [48] which is a hybrid model for large-scale optimization where maximum diminution of the problem is 200. But in our work, we struggle with very high dimension (more than 260,000) neural network and benchmark test problems, where the dimension is 1000.

CS can be applied to non-differentiable functions as well as discrete functions. Moreover, it can work properly on non-smooth functions as well as highly non-convex black-box functions. For this reason, it is less likely to be trapped in a local minimum in comparison with GD and CGD.

In terms of calculation, CS is computationally less intensive compared to other gradient-based methods because, in the CS algorithm, derivatives are not calculated. From sampling different points in search space, CS reaches to the target. So, this ability makes CS a good option for high-dimensional problems.

In a very high-dimensional problem, we can bundle the variables in a stochastic or deterministic way. This bundling approach helps us to change more variables in one step. We can call it the block coordinate search (BCS), and it is very similar to the BCD, but there is an important difference. In BCD, variables should be separated or partitioned in different groups, and for problems that are partially separable, we should use a meta-model or approximate model to separate variables or blocks of variables. In contrast, BCS does not need to separate variables in different partitions. We can group them and consider each group (or block) as one variable then we can change all variables in a block simultaneously while keeping other blocks fixed.

Initialization for each optimization algorithm is a very important phase. Some algorithms start with random initialization [49], [50], and some of them start with points that are considered as golden points [9]. But for large scale problems, the center point of the search space is the best point for sampling. Also, if the population-based algorithms are initialized, the best region for sampling is 60% of the center of the search space [51]. In the different schemes which are proposed in this work, we select the center point as starting point for each region or subregions.

In the next chapter, we proposed new schemes for CS-based algorithms and tried to enhance them. Moreover, we compared the results with other well-known algorithms to show the advantages of the proposed CS algorithms.

Chapter 3

Proposed CS Algorithms

In this chapter, we investigate different CS algorithms to develop them step by step for non-convex LSGO. We also target black-box optimization, so the algorithm can be used in many different applications and there is no need to calculate the fitness function. For black-box optimization, there is no mathematical proof for convergence. So, we try to empirically show the advantages of the proposed methods. There are not efficient methods for this domain of optimization, so we target it in this work. As mentioned in the previous chapter, because of the potential of the CS algorithms, we decided to enhance them for LSGO. There are a few numbers of approaches in this regard mentioned in the literature. MCS and GSS have some limitations, so they are not compatible with LSGO, as has been discussed before. During the exploration of proposing the most effective CS algorithm for high dimensional optimization, many methods have been tested, and among them, the best ones in terms of computational cost and ability to find the best solution are listed here. The schemes are started with a simplified CS algorithm, and step by step, more complicated schemes come to the scene, and eventually, we came with the best one in terms of performance and budget handling.

3.1 Simplifying Explanation of the Proposed Algorithms

Explaining the procedure in a way that is very clear and easy to understand is our goal in this section. We are working with a large number of variables, so it is very hard to imagine what happens during the optimization process. Therefore, we tried to explain this procedure with some images which represent the variables, box-constrains, and the way they are changed in the process. These changes are clearly shown in the images step by step. Figure 3.1 shows what each object and symbol in the images represents.

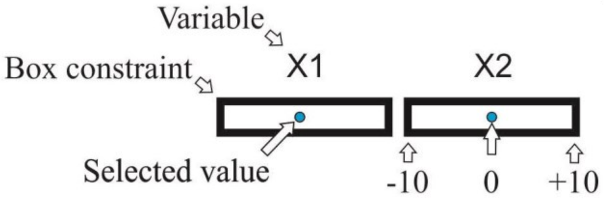


Figure 3.1: Definition of the objects which is used in other images

In these methods, we consider each dimension separately. Then for each dimension, some points are selected/sampled to evaluate the fitness function. Based on the value of the fitness function, a portion of related variable box-constraint has been selected, and remainder parts were removed. By this method, each variable (dimension) will shrink to the solution. We have tested tons of schemes, but in the following subsections, some of the best schemes are explained.

3.2 Conducting Comparative Study Using CEC-2013 Benchmark Problems

We need some benchmarks to empirically evaluate the performance of each scheme. By testing them on CEC-2013 benchmarks, which is for LSGO, the performance of each scheme can be evaluated. Table 3.1 shows the name, property, and box-constraint of this benchmark functions [52]. Dimension (D) of all functions is 1000. This benchmark is a minimization problem, and the minimum fitness value for each function is zero. All functions are rotated and shifted. The optimum values of the inputs are shifted in their box-constraint region. This makes the problem much harder. To have better analysis, we test the CS schemes by three budget levels, which are specified by NFC. We selected NFC equal to $20D$, $60D$ and $300D$ which represent low-budget, medium-budget and high-budget respectively. There is no specific definition or rule to show what amount of NFC can be considered as low, and what amount can be judged as high. Based on trial and error and a general understanding of optimization problems in the real world, we tried to push the limits of needed NFC to lower amounts in order to handle expensive optimization problems. It is noticeable that CS has a very high convergence rate among other algorithms, as mentioned above. Although we consider 300D as high-budget, in CEC-2013 for instance, the budget is 3000D, which is ten times more than the budget we used.

Table 3.1: CEC-2013 benchmark functions summary

Function	Function Name	Properties		Search Range
f1	Elliptic Function	Fully-separable Functions	Unimodal	$[-100, 100]^D$
f2	Rastrigin Function		Multimodal	$[-5, 5]^D$
f3	Ackley Function		Multimodal	$[-32, 32]^D$
f4	Elliptic Function	Functions with a separable subcomponent	Unimodal	$[-100, 100]^D$
f5	Rastrigin Function		Multimodal	$[-5, 5]^D$
f6	Ackley Function		Multimodal	$[-32, 32]^D$
f7	Schwefels Problem 1.2		Multimodal	$[-100, 100]^D$
f8	Elliptic Function	Functions with no separable subcomponents	Unimodal	$[-100, 100]^D$
f9	Rastrigin Function		Multimodal	$[-5, 5]^D$
f10	Ackley Function		Multimodal	$[-32, 32]^D$
f11	Schwefels Problem 1.2		Unimodal	$[-100, 100]^D$
f12	Rosenbrock's Function	Overlapping Functions	Multimodal	$[-100, 100]^D$
f13	Schwefels Function with Conforming Overlapping Subcomponents		Unimodal	$[-100, 100]^D$
f14	Schwefels Function with Conflicting Overlapping Subcomponents		Unimodal	$[-100, 100]^D$
f15	Schwefels Problem 1.2	Non-separable Functions	Unimodal	$[-100, 100]^D$

Most of the algorithms benefit from a random permutation. Therefore, these algorithms are stochastic. In order to have fair judgment for comparing each pair of CS, each algorithm ran 31 times, and then the average over 31 runs reported in the results. To make sure the results for comparison are significantly different or not, we used a t-test as a statistical test. The t-test, based on the mean value of the results, number of runs, and the standard deviation of the samples, shows that the result of two different stochastic algorithms are significantly different or not. In some cases, the results are different, but the t-test shows they are not significantly different. As a result, we considered them as the same results (tie).

3.3 Two-points CS

At first, we wanted to start with a simple scheme. In Two-points CS, two sample points for each variable are evaluated in each iteration. In Algorithm 1, pseudo-code of the two-points CS can be seen. The first variable range divided into two same-size parts (left and right). Then, the center point of each part was selected for function evaluation. For other variables, the center of their box-constraint has been considered for function evaluations. This process continued for other variables, one-by-one. Figure 3.2 shows these steps. In this figure, there are three variables, and the goal is to minimize the fitness function $f(\chi)$. When all variables divided in half, one iteration is completed, and then the second iteration starts. In this method, after each generation, box-constraint for each variable is halved. As a result, the box-constraints shrank by $2^{iteration}$ factor. So, the box-constraint shrinkage factor (BSF) can be defined as in Eq. (3.1).

$$BSF_{2-points} = (\tfrac{1}{2})^{iteration} \tag{3.1}$$

It is noticeable that the search space will shrink in an exponential way, which leads to fast convergence with a low number of fitness call that is very important in high-dimensional problems. The number of fitness call (NFC) can be calculated by Eq. (3.2) for each iteration, and in total iterations, NFC can be achieved by Eq. (3.3). In this equation, D stands for dimension or number of variables.

$$NFC_{Iteration} = 2D \tag{3.2}$$

$$NFC_{total} = 2D \times iteration \tag{3.3}$$

Algorithm 1 The pseudo-code of two-points CS algorithm

Input: $\varkappa\epsilon\mathbb{R}^D$: Solution vector, *D*: Dimension of the problem, *U*: upper bond, *L*: Lower bound, N_{ite}: Number of Iterations

Output: *S*: The best solution found so far

```
for $j \leftarrow 1$ to $N_{ite}$ do
    $C_i \leftarrow \frac{U_i - L_i}{2}$
    for $i \leftarrow 1$ to $D$ do
        $LS_i \leftarrow L_i + \frac{U_i - L_i}{4}$
        $RS_i \leftarrow L_i + 3 \times \frac{U_i - L_i}{4}$
        $S_{i_L} = f([C_1, C_2, ..., LS_i, ..., C_D])$
        $S_{i_R} = f([C_1, C_2, ..., RS_i, ..., C_D])$
        if $S_{i_L} < S_{i_R}$ then
            $U_i = U_i - \frac{U_i - L_i}{2}$
            $C_1 = LS_i$
            $S = S_{i_L}$
        end
        if $S_{i_L} > S_{i_R}$ then
            $L_i = L_i + \frac{U_i - L_i}{2}$
            $C_1 = RS_i$
            $S = S_{i_R}$
        end
    end
end
```

NFC has a severe effect on the computational cost of each optimization method. Therefore, schemes with less NFC are preferred. For comparing different approaches, we tried to keep the NFC the same to have a fair comparison. In this scheme, search space was cut after each iteration instead of just splitting them (one of the main differences between the proposed method and MSC). The main reason we preferred to cut the search space in LSGO is because there is no way to search all regions of the landscape. So, it was better to use the budget in an efficient way. Almost in all cases in LSGO, the budget (in terms of NFC) is limited. Therefore, we should neglect less important parts of the landscape and focus on more important parts. Cutting the less important areas based on sampling is the strategy in two-points CS, and it is not the best strategy, but it is effective based on imperial results.

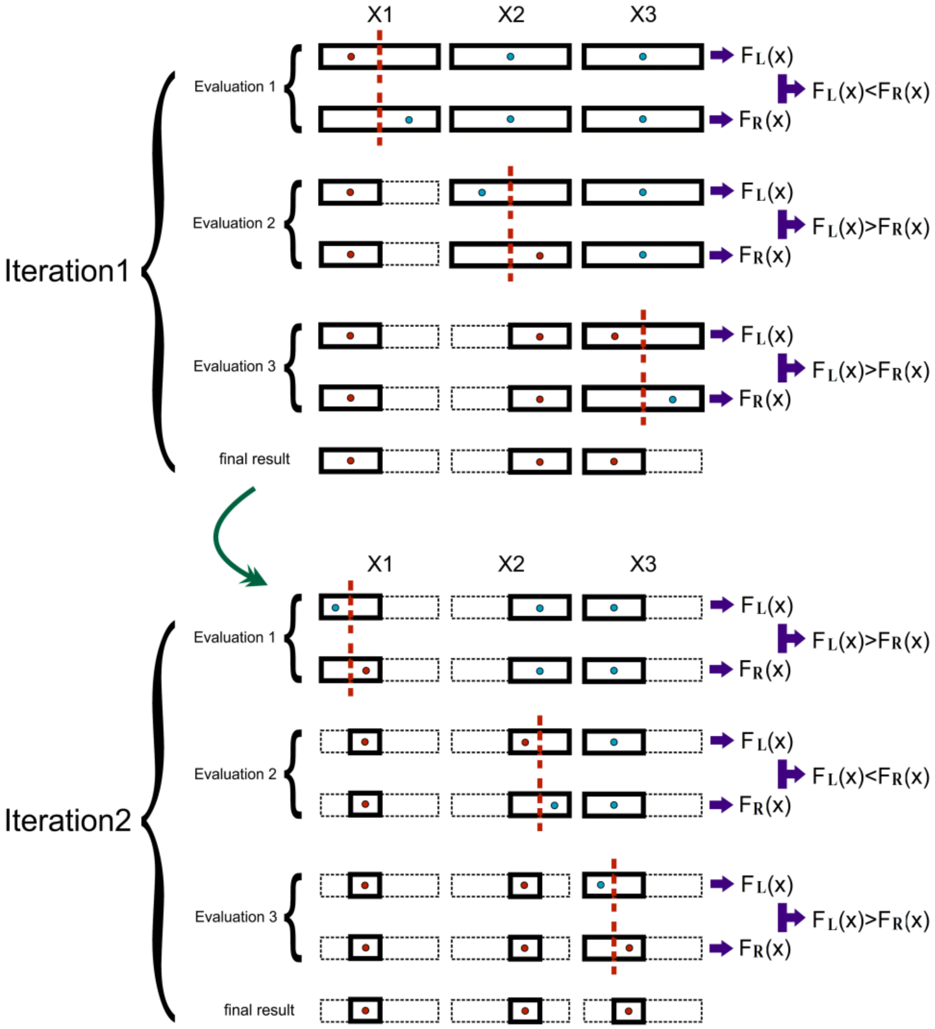


Figure 3.2: Procedure of search in two-points CS

Tables 3.2, 3.3, and 3.4 show comparative results of the two-point algorithm and other methods such as GD and CD as well as other coordinate search methods such as compass search and MCS. Results show the superior performance of our method on CEC-2013 LSGO benchmarks for low (20D), medium-budget (60D), and high-budget (300D).

Table 3.2: Comparing results of proposed CS with other algorithms, NFC=20D

D=1000 NFC = 20D Number of runs=31					
Function number	**2-points CS**	**GD**	**CD**	**Compass Search**	**MCS**
f1	843080.7	1.48E+11	1.48E+11	1.97E+11	1.59E+09
f2	2204.67	48792.86	48792.86	47482.77	2517.046
f3	20.52671	21.05481	21.04292	21.6933	20.25735
f4	2.53E+12	4.11E+13	4.1E+13	2E+13	2.03E+13
f5	14328626	29618104	29251297	45988175	15585165
f6	1063317	1084448	1081462	1068529	1068691
f7	9.15E+09	2.38E+13	2.36E+13	2.08E+14	2.31E+10
f8	1.40E+17	1.72E+18	1.65E+18	6.32E+17	1.20E+18
f9	1.2E+09	2.27E+09	1.88E+09	4.93E+09	1.12E+09
f10	96270265	99543697	98494030	96329132	96068867
f11	1.02E+12	8.01E+14	7.98E+14	8.99E+15	4.39E+12
f12	11863.3	1E+12	1E+12	1.69E+12	2.63E+08
f13	5.13E+10	2.51E+15	2.49E+15	4.89E+16	9.92E+10
f14	1.29E+12	1.71E+15	1.71E+15	4.79E+16	2.33E+12
f15	2.69E+08	5.49E+14	5.49E+14	2.25E+15	4.03E+10

Table 3.3: Comparing results of proposed CS with other algorithms, NFC=60D

D=1000 NFC = 60D Number of runs=31					
Function number	**2-point CS**	**GD**	**CD**	**Compass Search**	**MCS**
f1	325885.6	6.75E+10	6.75E+10	2.06E+11	1.55E+09
f2	2172.363	48792.86	48792.86	47277.96	2204.238
f3	20.00344	20.67382	20.67625	21.67354	20.05479
f4	2.66E+12	3.63E+12	3.32E+12	4.98E+13	3.7E+12
f5	14805066	31416388	27843279	44997464	14020741
f6	1053955	1086606	1080990	1068907	1055805
f7	8.6E+09	1.32E+11	1.31E+11	1.09E+14	1.94E+10
f8	1.39E+17	2.57E+17	2.22E+17	3.95E+17	2.7E+17
f9	1.16E+09	1.91E+09	1.92E+09	4.75E+09	1.25E+09
f10	95253252	98858435	97845643	94916801	94857014
f11	1.09E+12	1.44E+13	1.3E+13	4.62E+15	4.26E+12
f12	9612.465	2.78E+11	2.78E+11	1.69E+12	2.66E+08
f13	5.51E+10	2.75E+13	2.7E+13	8.27E+16	1.44E+11
f14	1.2E+12	1.67E+12	1.3E+12	7.32E+15	2.06E+12
f15	3.38E+08	2.32E+13	2.32E+13	2.39E+15	5.77E+10

Table 3.4: Comparing results of proposed CS with other algorithms, NFC=300D

D=1000 NFC = 300D Number of runs=31					
Function number	**2-point CS**	**GD**	**CD**	**Compass Search**	**MCS**
f1	325885.4	2.28E+09	2.28E+09	2.02E+11	1.55E+09
f2	2172.363	48792.86	48792.86	47274.97	2204.238
f3	20.00344	20.57917	20.58149	21.66958	20.05479
f4	2.1E+12	7.85E+11	1.5E+11	2.1E+13	1.02E+13
f5	13011859	25420655	25051756	44218103	13218538
f6	1053729	1074855	1078828	1069509	1062824
f7	7.45E+09	2.96E+09	1.14E+09	2.69E+14	2.42E+10
f8	1.08E+17	3.49E+16	6.13E+15	7.14E+17	3.53E+17
f9	1.42E+09	1.71E+09	1.77E+09	4.3E+09	1.2E+09
f10	95320460	99319298	98749946	96503072	95096945
f11	1.03E+12	5.14E+11	1.25E+11	6.62E+15	2.13E+12
f12	9215.382	1.55E+09	1.55E+09	1.68E+12	2.71E+08
f13	6.1E+10	1.67E+10	5.24E+09	3.35E+16	2.4E+11
f14	1.07E+12	6.48E+11	1.03E+11	4.34E+17	3.02E+12
f15	2.34E+08	3.25E+09	2.45E+08	1.83E+15	5.14E+10

Based on the experimental results for low and medium-budget, two-points CS shows superior performance in the vast majority of the benchmark functions. The second best is MCS, but on the high-budget condition. Two-points also win in a simple majority of functions (all non-convex functions except ƒ7), but CD is the second-best, and win mostly in convex (unimodal) functions. Results show two-points CS performs better than other algorithms for non-convex optimization, so, proposed CS algorithm for non-convex optimization is a good choice among other options.

3.3 Three-points CS

In order to go one step further, we realized that in each step of two-points CS, two samples of each variable should be evaluated, but also the winner of the previous step is the center point of the current step. Therefore, without any extra fitness calls (except for the first step that needs one extra call), we have access to three points for each step, and with more points, the chance of finding a better solution will be increased. Left, center, and right points can be compared based on their fitness values, and the best one will be selected as the center of the next step. Algorithm 2 shows the pseudo-code of this method. For example, if the right or left point has the best value, it will be just like two-points CS. But if the center point has the best value, the center part of the box-constraint will be selected, and 25% of the left and right will be cut.

Figure 3.3 illustrates the process of the three-points CS step by step by three variables. In this scheme, the BSF can be calculated by Eq. (3.4), which is exactly the same as two-point CS. Total NFC can be obtained by Eq. (3.5), which needs one fitness to call more than two-point CS, which is negligible in high dimension.

$$BSF_{3-point} = (\frac{1}{2})^{iteration} \tag{3.4}$$

$$NFC_{total} = 2D \times iteration + 1 \tag{3.5}$$

Algorithm 2 The pseudo-code of three-points CS algorithm

Input: $\varkappa \epsilon \mathbb{R}^D$: Solution vector, *D*: Dimension of the problem, *U*: upper bond, *L*: Lower bound, N_{ite}: Number of Iterations

Output: *S*: The best solution found so far

```
for $j \leftarrow 1$ to $N_{ite}$ do
    $C_i \leftarrow \frac{U_i - L_i}{2}$
    for $i \leftarrow 1$ to $D$ do
        $LS_i \leftarrow L_i + \frac{U_i - L_i}{4}$
        $RS_i \leftarrow L_i + 3 \times \frac{U_i - L_i}{4}$
        $S_{i_L} = f([C_1, C_2, ..., LS_i, ..., C_D])$
        $S_{i_R} = f([C_1, C_2, ..., RS_i, ..., C_D])$
        if $S_{i_L} < S_{i_R}$ then
            if $S_{i_L} < S_{i-1}$ then
                $U_i = U_i - \frac{U_i - L_i}{2}$
                $C_i = LS_i$
                $S = S_{i_L}$
            Else
                $U_i = U_i - \frac{U_i - L_i}{4}$ and $L_i = L_i + \frac{U_i - L_i}{4}$
        end
        if $S_{i_R} < S_{i_L}$ then
            if $S_{i_R} < S_{i-1}$ then
                $L_i = L_i + \frac{U_i - L_i}{2}$
                $C_i = RS_i$
                $S = S_{i_R}$
            Else
                $U_i = U_i - \frac{U_i - L_i}{4}$ and $L_i = L_i + \frac{U_i - L_i}{4}$
        end
    end
end
```

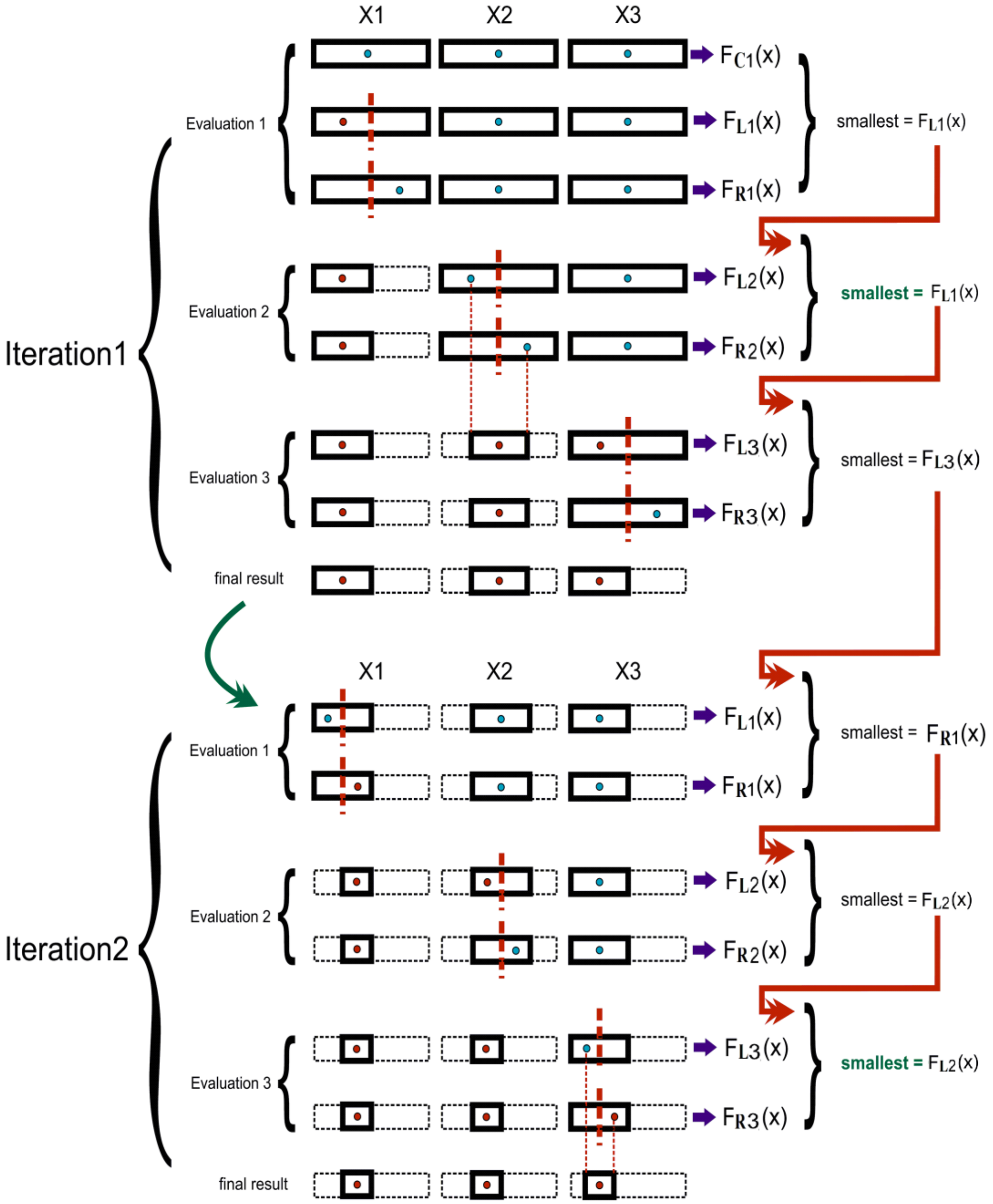


Figure 3.3: Procedure of search in three-points CS

Tables 3.5 and 3.6 show the results of three-points CS and two-points CS over benchmarks for low and medium-budget, respectively. Figure 3.4 illustrates the performance plots of two-points and three-points CS.

Table 3.5: Comparing three-points with two-points CS, NFC=20D

D=1000 NFC = 20D Number of runs=31			
Function	**(a): 3-points CS**	**(b): 2-points CS**	**Ratio (a)/(b)**
f1	55588.5	843080.7	0.07
f2	1848.478	2204.67	0.84
f3	20.02986	20.52671	0.98
f4	1.32E+12	2.53E+12	0.52
f5	18423850	14328626	1.29
f6	1051798	1063317	0.99
f7	6.2E+09	9.15E+09	0.68
f8	8.09E+16	1.40E+17	0.58
f9	1.39E+09	1.2E+09	1.08
f10	94715529	96270265	0.98
f11	9E+11	1.12E+12	0.88
f12	7454.659	11863.3	0.63
f13	2.61E+10	5.13E+10	0.51
f14	6.27E+11	1.29E+12	0.49
f15	6.61E+08	2.69E+08	2.46
t-test results	**Win = 11**	**Win = 2**	**AVG=0.86**
	Tie = 2		

Table 3.6: Comparing three-points with two-points CS, NFC=60D

D=1000 NFC = 60D Number of runs=31			
Function	**(a): 3-points CS**	**(b): 2-points CS**	**Ratio (a)/(b)**
f1	5.84E-08	325885.6	0
f2	1846.641	2172.363	0.85
f3	20	20.00344	1
f4	1.18E+12	2.66E+12	0.44
f5	18144256	14805066	1.23
f6	1045799	1053955	0.99
f7	4.82E+09	8.6E+09	0.56
f8	8.90E+16	1.39E+17	0.64
f9	1.35E+09	1.16E+09	1.16
f10	94132474	95253252	0.99
f11	7.62E+11	1.09E+12	0.7
f12	6470.146	9612.465	0.67
f13	2.31E+10	5.51E+10	0.42
f14	5.24E+11	1.2E+12	0.44
f15	6.65E+08	3.38E+08	1.97
t-test results	**Win = 12**	**Win = 3**	**AVG=0.80**
	Tie = 0		

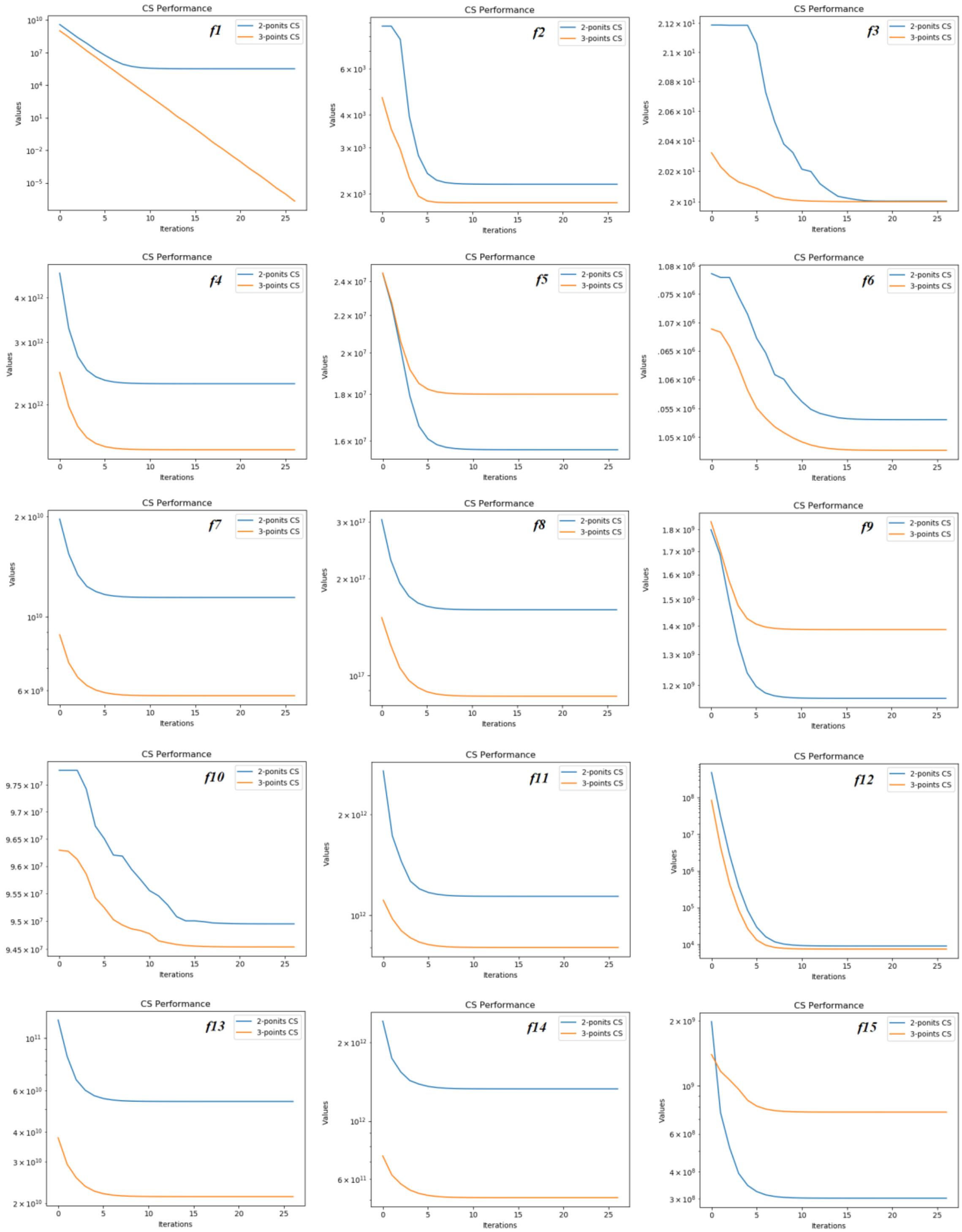


Figure 3.4: Performance plots of two-points and three -points CS scheme

Based on the t-test results, the three-points scheme outperforms two-points CS in the vast majority of benchmarks. It shows when the center point is considered, it is more likely to find a better solution. Moreover, performance plots show that in most cases, three-points CS starts from a better point, which results in better performance. Besides, three-points CS keeps the best solution found so far, so there is no fluctuation in its performance plot, which is not the case for two-points CS. Three-points CS can keep the right, left, or center part of the box-constraint in each iteration, so the intersection of the left side and right side also is considered. Both of these algorithms converge very fast, so the results for high-budget (300D) will be the same as the medium-budget.

Both Two-points and Three-points CS have an origin in binary-search method, this why when the landscape is monotonically increasing/decreasing the schemes works perfectly.

3.4 Overlapped two-points CS

After some experiments and investigations, we found out that cutting half of each variable in each iteration can damage the overall performance of the optimization method. In other words, by using very few samples, each variable is cut in half in each iteration, and the search space shrinks by the rate of 2^D per iteration. So, the chance of losing the region of interest in a search space is high. To address this problem, the variables are cut in a gentle way. Therefore, in this scheme, for each iteration, less than half of each variable is cut. We call it overlapped because the left side and right side of the box-constraint have overlapped with each other.

There is a trade-off here between BSF and NFC. This algorithm can find a better result, but it needs more fitness calls, which are not desirable.

This scheme is like two-points CS, but instead of choosing the right half or the left half, more than 50% of the right or left part is selected in each step. For example, if the right point has the best value, right three-quarters of the box-constraint will be kept, and one quarter will be cut. In this scheme, the selected portion of the box-constraint (SPB) can be between 0.5 to one (Eq. 3.6). 0.5 means 50 percent of the box-constraint, and one means 100 percent of the box-constraint.

$$0.5 < SPB < 1 \tag{3.6}$$

In Figure 3.5, this method has been illustrated for three variables and SPB =75%. BSF can be calculated by Eq. (3.7), and it shows more iterations are needed to have specific shrinkage than two-point CS. Total NFC can be obtained by Eq. (3.8).

$$BSF_{2-point-overlapped} = SPB^{iteration} \tag{3.7}$$

$$NFC_{total} = 2D \times iteration \tag{3.8}$$

Experimental tests on CEC-2013 benchmarks showed that if SPB increases, on one side, the ability of the algorithm to find the better solution will be increased. But on the other side, the number of iterations which is needed to find the solution will also increase. This means more budget in terms of NFC is required to address the optimization problem. In some cases, finding a better solution has a higher priority, so a high amount of SPB can be selected. The best value for SPB depends on the problem and budget, and we selected SPB=0.9 with trial and error on the benchmark functions with specific NFC. SPB = 0.9 means in each iteration, 90% of the box-constraint will remain, and 10% of that will be removed. This 90%-overlapped scheme shows outstanding performance in terms of finding a better solution.

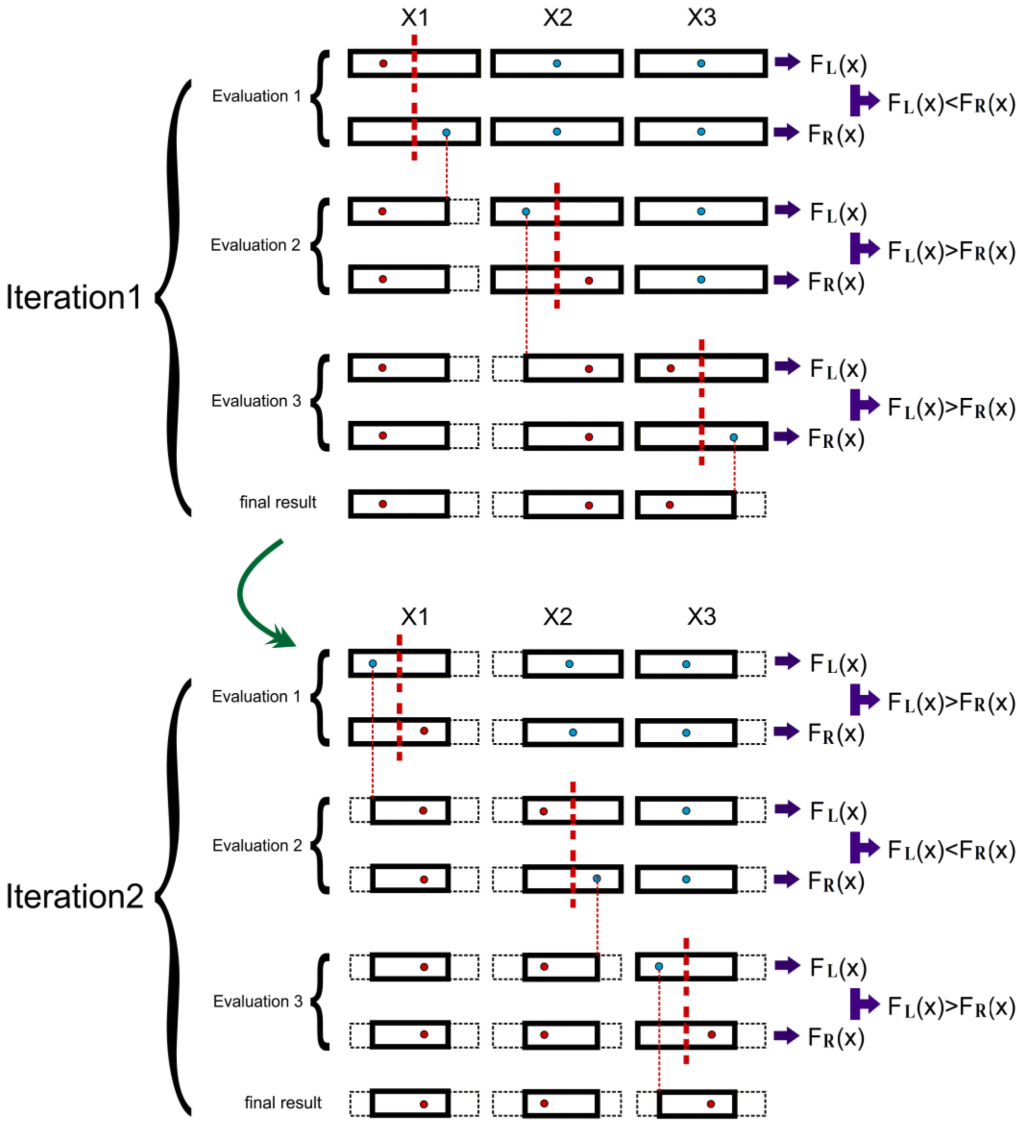


Figure 3.5: Procedure of search in two-points overlapped CS, SPB=0.75

Tables 3.7, 3.8, 3.9 reported the results of overlapped two-points CS and three-points CS in low, medium, and high-budget situations.

Table 3.7: Comparing three-points with overlapped two-points CS, NFC=20D

D=1000 NFC = 20D Number of runs=31			
Function	**(a): 3-points CS**	**(b): Overlapped 2-points CS SPB=0.90**	**Ratio (a)/(b)**
ƒ1	55588.5	1.73E+10	0
ƒ2	1848.478	8520.449	0.22
ƒ3	20.02986	21.16595	0.95
ƒ4	1.32E+12	3.81E+12	0.35
ƒ5	18423850	19578372	0.94
ƒ6	1051798	1079268	0.97
ƒ7	6.2E+09	1.01E+10	0.61
ƒ8	8.09E+16	2.31E+17	0.35
ƒ9	1.29E+09	1.42E+09	0.91
ƒ10	94715529	97867079	0.97
ƒ11	9E+11	1.42E+12	0.63
ƒ12	7454.659	7.07E+09	0
ƒ13	2.61E+10	6.79E+10	0.38
ƒ14	6.27E+11	1.44E+12	0.43
ƒ15	6.61E+08	5.92E+08	1.12
t-test results	**Win = 13**	**Win = 0**	**AVG=0.59**
	Tie = 2		

Table 3.8: Comparing three-points with overlapped two-points CS, NFC=60D

D=1000 NFC = 60D Number of runs=31			
Function	**(a): 3-points CS**	**(b): Overlapped 2-points CS SPB=0.90**	**Ratio (a)/(b)**
ƒ1	5.84E-08	3E+08	0
ƒ2	1846.641	5153.876	0.36
ƒ3	20	20.98465	0.95
ƒ4	1.18E+12	1.07E+12	1.1
ƒ5	18144256	10176286	1.78
ƒ6	1045799	1077254	0.97
ƒ7	4.82E+09	3.5E+09	1.38
ƒ8	8.90E+16	4.03E+16	2.21
ƒ9	1.35E+09	7.31E+08	1.85
ƒ10	94132474	97678093	0.96
ƒ11	7.62E+11	3.46E+11	2.2
ƒ12	6470.146	2384413	0
ƒ13	2.31E+10	1.75E+10	1.32
ƒ14	5.24E+11	5.48E+11	0.96
ƒ15	6.65E+08	95282892	6.98
t-test results	**Win = 6**	**Win = 7**	**AVG=1.53**
	Tie = 2		

Table 3.9: Comparing three-points with overlapped two-points CS, NFC=300D

D=1000 NFC = 300D Number of runs=31			
Function	**(a): 3-points CS**	**(b): Overlapped 2-points CS SPB=0.90**	**Ratio (a)/(b)**
f1	0	0.00321	0
f2	1846.641	123.375	14.29
f3	20	20	1
f4	1.34E+12	8.8E+11	1.52
f5	18085987	5851969	3.13
f6	1046786	1027619	1.02
f7	4.66E+09	3.1E+09	1.52
f8	7.68E+16	3.52E+16	2.17
f9	1.45E+09	4.5E+08	3.23
f10	94275973	9.3E+07	1.02
f11	8.26E+11	2.7E+11	3.03
f12	7529.469	1252.24	5.88
f13	2.38E+10	1.5E+10	1.61
f14	5.94E+11	4.3E+11	1.37
f15	7.3E+08	5.9E+07	12.5
t-test results	**Win = 1**	**Win = 13**	**AVG=3.55**
	Tie= 1		

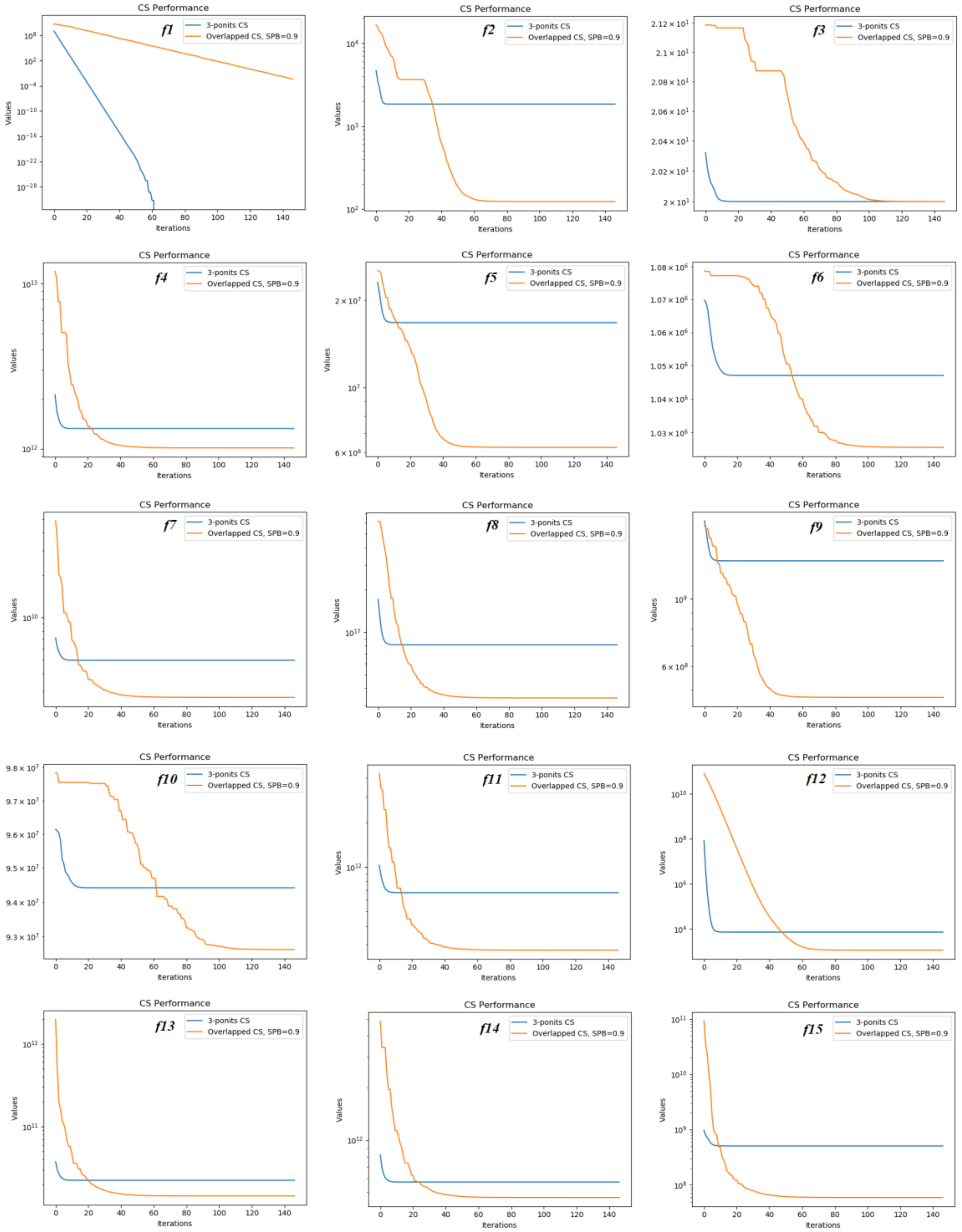


Figure 3.6: Performance plot of three-points CS and Overlapped two-points CS (SPB = 0.9)

Based on the empirical results, for low-budget (20D) situations, three-points CS performs the best, and wins on 13 benchmarks. But by increasing NFC results reversed, and the overlapped method shows its advantages. For NFC=300D, it wins on 13 benchmarks. This was expected because the overlapped optimizer gradually moves toward the solution. In contrast, three-points CS can converge very fast, but it is more likely to lose the region of interest on the landscape. Performance plots also illustrate that there are cross points in the vast majority of performance plots. This confirms that three-points CS is better in low-budget situations, and overlapped CS performs better in high-budget situations.

After analyzing 90%-overlapped CS, we came to this conclusion that instead of sampling one point from 90% of box-constraint, which will be preserved, the sample can be taken from 10% of the box-constraint that should be removed. We assume that this sampling method can better represent the smaller part. Then we took it one step further, and instead of sampling the center point of the 10% part of both ends, the extreme points of each box-constraint have been considered as sample points. Algorithm 3 shows the procedure. This change results in superior performance in terms of finding better solutions. Figure 3.7 shows this transition towards two-extreme points CS from two-points CS. Figure 3.8 illustrates how the two-extreme points scheme works.

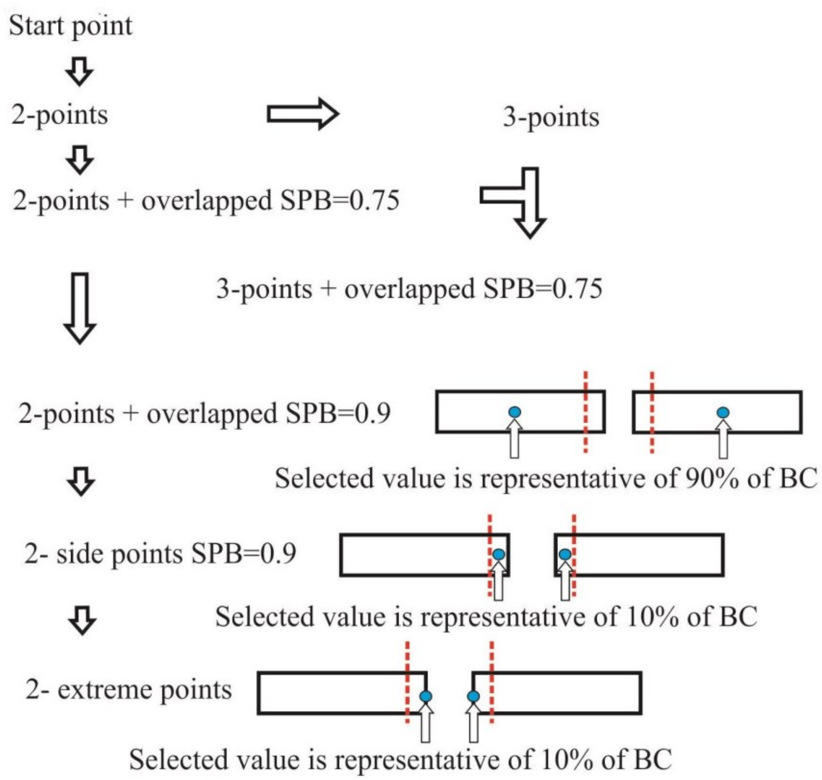

Start point
2-points
3-points
2-points + overlapped SPB=0.75
3-points + overlapped SPB=0.75
2-points + overlapped SPB=0.9
Selected value is representative of 90% of BC
2- side points SPB=0.9
Selected value is representative of 10% of BC
2- extreme points
Selected value is representative of 10% of BC

Figure 3.7: Transition from two-points CS to two-extreme points CS

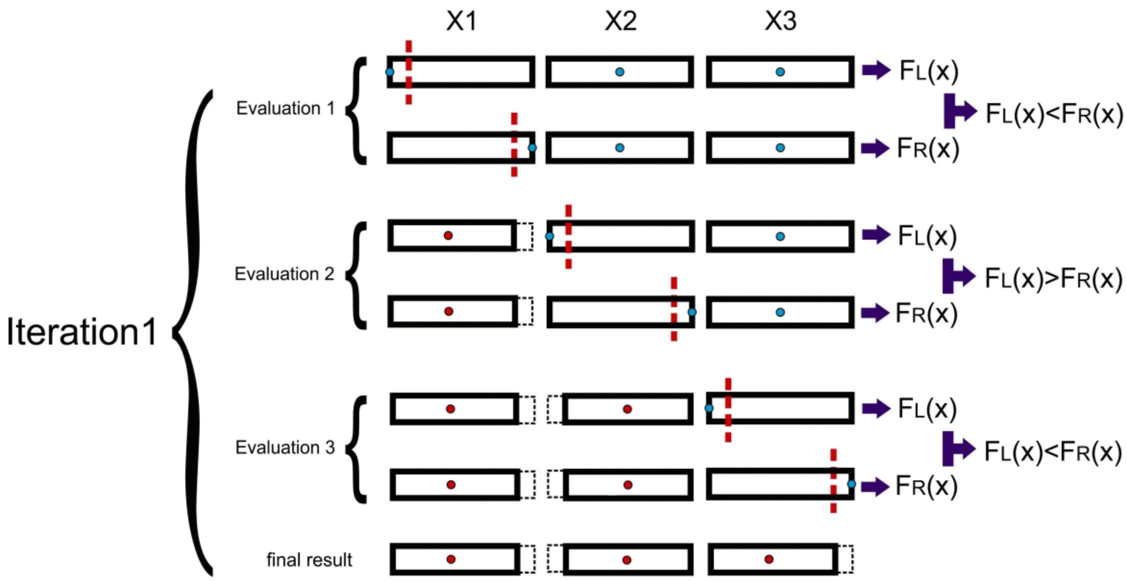


Figure 3.8: Two-extreme points scheme procedure

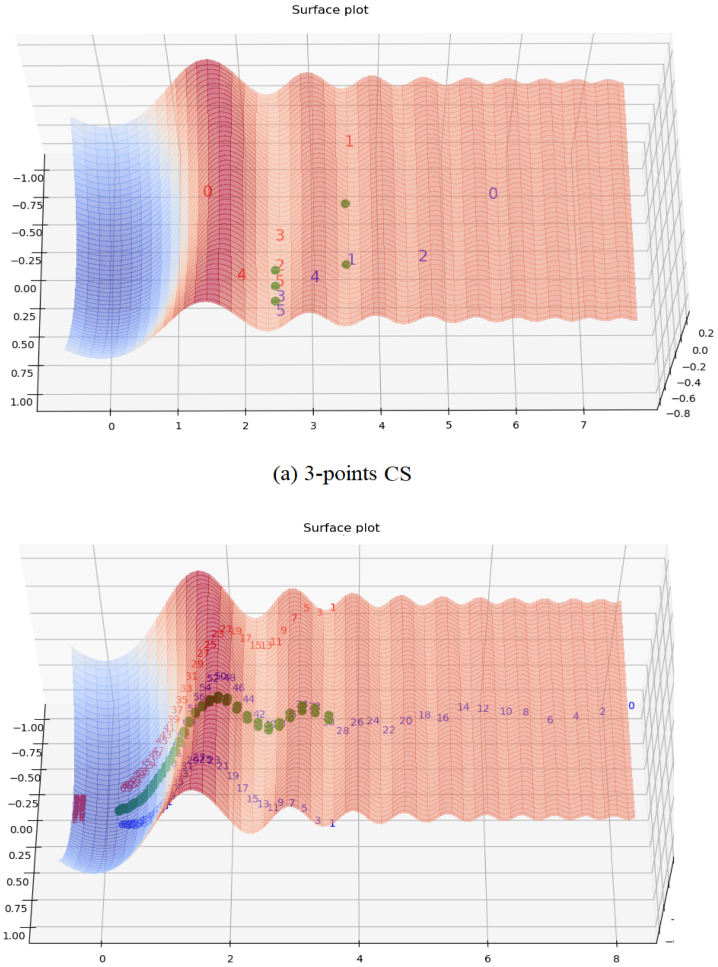
Algorithm 3 The pseudo-code of two- extreme points CS algorithm

Input: $\varkappa \epsilon \mathbb{R}^D$: Solution vector, *D*: Dimension of the problem, *U*: upper bond, *L*: Lower bound, N_{ite}: Number of Iterations, *BSF*: Box-constraint shrinkage factor

Output: *S*: The best solution found so far

```
for $j \leftarrow 1$ to $N_{ite}$ do
    $C_i \leftarrow \frac{U_i - L_i}{2}$
    for $i \leftarrow 1$ to $D$ do
        $LS_i \leftarrow L_i$
        $RS_i \leftarrow U_i$
        $S_{i_L} = f([C_1, C_2, \ldots, LS_i, \ldots, C_D])$
        $S_{i_R} = f([C_1, C_2, \ldots, RS_i, \ldots, C_D])$
        if $S_{i_L} < S_{i_R}$ then
            $U_i = U_i - (U_i - L_i) \times (1 - BSF)$
            $C_1 = \frac{(L_i - U_i)}{2}$
            $S = S_{i_L}$
        end
        if $S_{i_L} > S_{i_R}$ then
            $L_i = L_i + (U_i - L_i) \times (1 - BSF)$
            $C_1 = \frac{(L_i - U_i)}{2}$
            $S = S_{i_R}$
        end
    end
end
```

Figure 3.9 illustrates the procedure and steps for three-points and two-extreme points CS. Blue numbers show the right side, and the red ones show the left side of the box-constraint. Numbers also show the evaluation number, which is illustrated in Figure 3.6. Green dots show the final result of each iteration. Three-points can pass some modals on the landscape, but cannot find the global solution. In contrast, the two-extreme points CS can find the pass towards the global optimum while passing all local minimums.



(a) 3-points CS

(b) 2-extreme points CS

Figure 3.9: Visualization of three-points (a) and two-extreme points CS (b), Red and blue numbers: left side, and right side of the box-constraint respectively, Green dots: Solutions, Numbers: Evaluations

Tables 3.10 and 3.11 show the difference between the results of overlapped two-points and two-side points CS for medium (60D) and high-budget (300D). Results show that the overlapped two-side points as we expedited, works way better than the other one and remarkably wins in almost all benchmarks. It shows, by a limited number of sample points, eliminating a small part which contains a bad solution is much better than keeping the large part of the box-constraint, which contains a good solution.

Table 3.10: Comparing overlapped two-points with two-side points CS, NFC=60D

D=1000 NFC = 60D Number of runs=31			1
Function	**(a): Overlapped 2-points CS-SPB=0.90**	**(b): Overlapped 2-side points CS-SPB=0.90**	**Ratio (a)/(b)**
ƒ1	3E+08	12313212	24.37
ƒ2	5153.876	12584.74	0.41
ƒ3	20.98465	21.72772	0.97
ƒ4	1.07E+12	2.6E+11	4.11
ƒ5	10176286	4330597	2.35
ƒ6	1077254	1063627	1.01
ƒ7	3.5E+09	1.44E+09	2.44
ƒ8	4.03E+16	1.11E+16	3.62
ƒ9	7.31E+08	3.47E+08	2.11
ƒ10	97678093	97838796	1
ƒ11	3.46E+11	1.49E+11	2.32
ƒ12	2384413	272820.7	8.74
ƒ13	1.75E+10	9.9E+09	1.77
ƒ14	5.48E+11	3.81E+11	1.44
ƒ15	95282892	35369667	2.69
t-test results	**Win = 2**	**Win = 12**	**AVG=3.95**
	Tie = 1		

Table 3.11: Comparing overlapped two-points with two-side points CS, NFC=300D

D=1000 NFC = 300D Number of runs=31			
Function	**(a): Overlapped 2-points CS-SPB=0.90**	**(b): Overlapped 2-side points CS-SPB=0.90**	**Ratio (a)/(b)**
ƒ1	0.00321	6.20E-05	51.73
ƒ2	123.375	3.08E-09	4E+10
ƒ3	20.00012	20	1
ƒ4	8.8E+11	1.8E+11	4.99
ƒ5	5851969	4089007	1.43
ƒ6	1027619	1001150	1.03
ƒ7	3.1E+09	1E+09	2.93
ƒ8	3.52E+16	8.7E+15	4.06
ƒ9	4.5E+08	3.3E+08	1.34
ƒ10	9.3E+07	9.1E+07	1.02
ƒ11	2.7E+11	1.1E+11	2.57
ƒ12	1252.24	1044.59	1.2
ƒ13	1.5E+10	7.9E+09	1.88
ƒ14	4.3E+11	3.3E+11	1.3
ƒ15	5.9E+07	2.2E+07	2.67
t-test results	**Win = 0**	**Win = 15**	**AVG=5.27** (ƒ2 excluded)
	Tie = 0		

Tables 3.11 and 3.12 reports the results of medium and high-budget on overlapped two-side points and two-extreme points scheme, and shows that two-extreme points CS works slightly better than the other one in terms of results ratio and the number of winners.

So, based on numerical results, and for the sake of simplicity, we selected two-extreme points CS as the best scheme so far for medium and high-budget optimization.

Table 3.12: Overlapped two-side points vs. two-extreme points CS, NFC=60D

D=1000 NFC = 60D Number of runs=31			1
Function	**(a): Overlapped 2-side points CS-SPB=0.90**	**(b): Overlapped 2-extreme points CS-SPB=0.90**	**Ratio (a)/(b)**
f1	12313212	11608004	1.06
f2	12584.74	14546.02	0.87
f3	21.72772	21.73485	1
f4	2.6E+11	2.45E+11	1.06
f5	4330597	4081267	1.06
f6	1063627	1064437	1
f7	1.44E+09	1.41E+09	1.02
f8	1.11E+16	1.18E+16	0.94
f9	3.47E+08	3.08E+08	1.13
f10	97838796	98075478	1
f11	1.49E+11	1.58E+11	0.94
f12	272820.7	292186.5	0.93
f13	9.9E+09	1.07E+10	0.93
f14	3.81E+11	4.06E+11	0.94
f15	35369667	32700678	1.08
t-test results	**Win = 3**	**Win = 3**	**AVG=0.997**
	Tie = 9		

Table 3.13: Overlapped two-side points vs. two-extreme points CS, NFC=300D

D=1000 NFC = 300D Number of runs=31			1
Function	**(a): Overlapped 2-side points CS-SPB=0.90**	**(b): Overlapped 2-extreme points CS-SPB=0.90**	**Ratio (a)/(b)**
f1	6.20E-05	3.95E-05	1.57
f2	3.08E-09	28.5836	0
f3	20	20.0033	1
f4	1.8E+11	2E+11	0.86
f5	4089007	3777400	1.08
f6	1001150	1000336	1
f7	1E+09	1.1E+09	0.92
f8	8.7E+15	8.3E+15	1.05
f9	3.3E+08	3E+08	1.1
f10	9.1E+07	9.1E+07	1
f11	1.1E+11	1.3E+11	0.8
f12	1044.59	1044.28	1
f13	7.9E+09	8.3E+09	0.94
f14	3.3E+11	3.4E+11	0.97
f15	2.2E+07	2.1E+07	1.04
t-test results	**Win = 3**	**Win = 5**	**AVG=0.955**
	Tie= 7		

3.5 Complimentary Techniques

Each of the above schemes can be more comprehensive by adding the following complimentary options to them. But each of these techniques has advantages and disadvantages.

3.5.1 Investigating Effects of Permutation

As mentioned in Chapter 2, the CD can benefit from random permutation to support a better exploration. We can take advantage of this technique in CS methods as well, but it has its own pros and cons. In each iteration, the order of the variables (permutation) can be fixed or varied. If the order of variables is fixed, the optimizer will work in a deterministic way. This means that if the optimization algorithm runs ten times, for instance, every time the output will be the same. Another approach is that for the first iteration, the order of variables can be shuffled, and for the rest iterations, orders will not change. So, each run will start with a different permutation. Moreover, for each iteration, the order can be shuffled as well. So, for each iteration, different permutations can exist. In these situations, the optimizer works in a stochastic manner, and every time the optimizer is called, it will come up with a different result. But the chance of finding a superior solution in this situation also increases.

Table 3.14: Permutation effect, NFC=20D

D=1000 NFC = 20D Number of runs=31			
Permutation	**In all iterations**	**Just in the first iteration**	
Function	**(a): 3-points CS**	**(b): 3-points CS**	**Ratio (a)/(b)**
f1	55588.5	55588.5	1
f2	1848.478	1848.478	1
f3	20.02986	20.02986	1
f4	1.32E+12	1.38E+12	0.96
f5	18423850	18820507	0.98
f6	1051798	1050825	1
f7	6.2E+09	5.79E+09	1.07
f8	8.09E+16	8.57E+16	0.94
f9	1.29E+09	1.4E+09	0.92
f10	94715529	94827625	1
f11	9E+11	7.05E+11	1.28
f12	7454.659	6567.581	1.14
f13	2.61E+10	2.52E+10	1.04
f14	6.27E+11	5.33E+11	1.18
f15	6.61E+08	6.95E+08	0.95
t-test results	**Win = 1**	**Win = 1**	**AVG=1.03**
	Tie = 13		

Table 3.15: Permutation effect, NFC=60D

D=1000 NFC = 60D Number of runs=31			
Permutation	**In all iterations**	**Just in the first iteration**	
Function	**(a): 3-points CS**	**(b): 3-points CS**	**Ratio (a)/(b)**
f1	5.84E-08	5.84E-08	1
f2	1846.641	1846.641	1
f3	20	20	1
f4	1.18E+12	1.5E+12	0.79
f5	18144256	17298552	1.05
f6	1045799	1046763	1
f7	4.82E+09	5.47E+09	0.88
f8	8.90E+16	8.42E+16	1.06
f9	1.35E+09	1.38E+09	0.98
f10	94132474	94366421	1
f11	7.62E+11	5.55E+11	1.37
f12	6470.146	5427.735	1.19
f13	2.31E+10	2.54E+10	0.91
f14	5.24E+11	5.87E+11	0.89
f15	6.65E+08	4.67E+08	1.42
t-test results	**Win = 3**	**Win = 2**	**AVG=1.036**
	Tie = 10		

Tables 3.14 and 3.15 show the difference between different permutations in the first and all iterations for the low and medium-budget. Results show a slight difference between them, but in terms of ratio, having a permutation in all iterations makes three percent improvements. There is one exception here. In the case of fully separable problems (*f1* to *f3*), the order of the variables is not important. Therefore, having a different permutation will not affect the results, and it is indifference.

3.5.2 Re-evaluating in Iteration

In previous methods, in each iteration, we start from the first variable, and select a portion of it, then select the second one, and repeat this procedure until the last variables (assume there is no permutation). But there is a very important point here. When we want to decide which portion of the first variable should be kept, other variables have the center point value. However, when we have reached the last variables, other first variables have their new values. To have a fair situation for the first variables, we can re-evaluate them when new values for other variables are selected (end of iteration). So, we can re-evaluate the selected portion of each variable again, with the hope of having a better decision. Figure 3.10 illustrates this method for three-points CS. In this example, in the first iteration, one extra set of evaluations has been performed, and as a result, the first iteration ended with a slightly different result for variable one. Without re-evaluating, the left part of variable one has been selected. However, with the re-evaluation, the right part has been preserved.

Re-evaluating can be performed one or more times in either the first or every iteration. It can be helpful in finding a better solution. The downside is that each re-evaluation needs more fitness calls, which is undesirable. As Figure 3.10 shows, in one iteration, the re-evaluating procedure needs two times (if reevaluation is performed one time) more fitness calls, but in some cases, the final result is better. Re-evaluating for overlapped methods with SPB=0.9 does not make noticeable changes in results because, in that method, the search space is cut slowly, and re-evaluating will not relatively change the results.

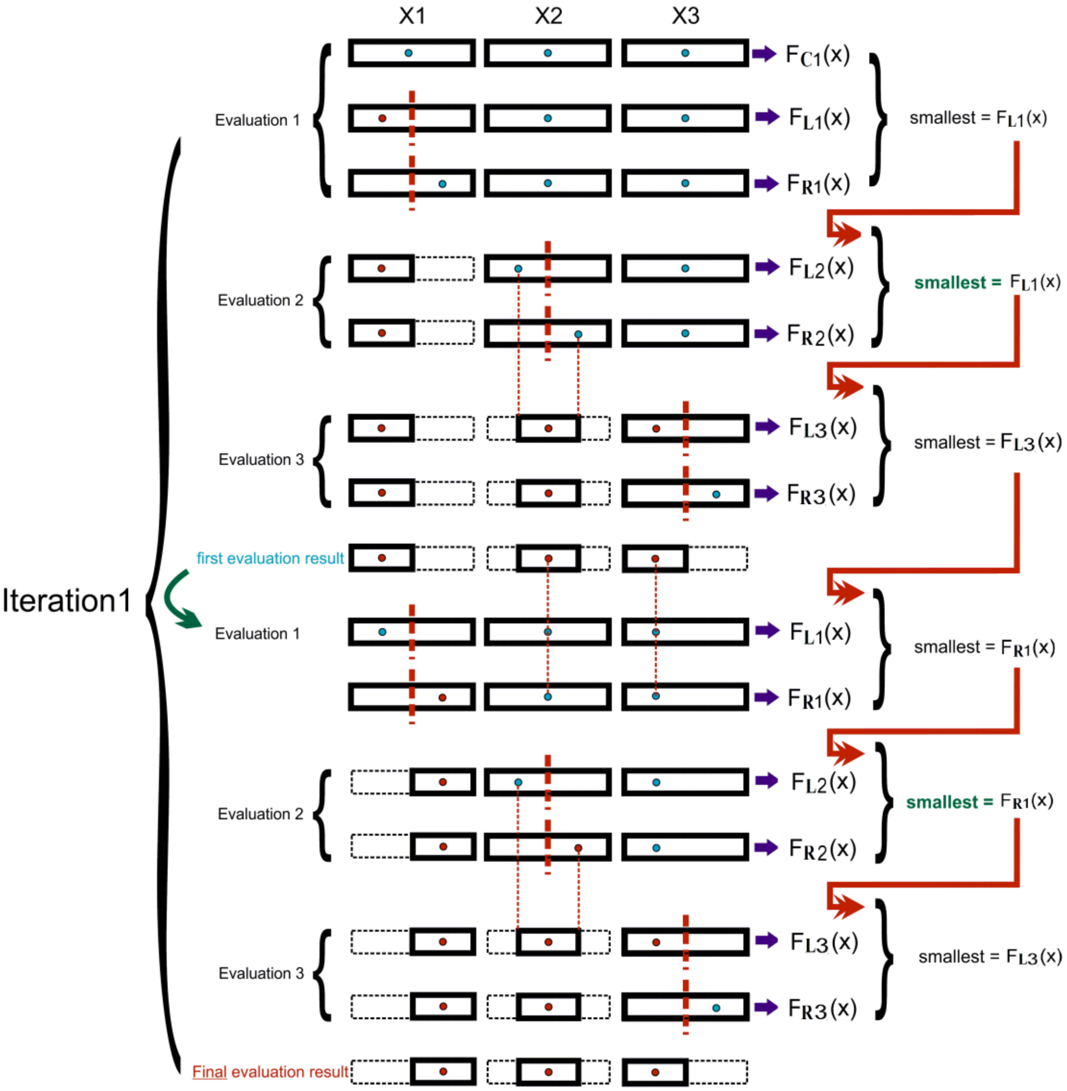


Figure 3.10: Re-evaluating procedure in CS

Table 3.16: Effect of re-evaluating, NFC=20D

D=1000 NFC = 20D Number of runs=31			1
Re-evaluating	No	Yes	
Function	(a): 3-points CS	(b): 3-points CS	Ratio (a)/(b)
ƒ1	55588.5	228367.9	0.24
ƒ2	1848.478	1854.092	1
ƒ3	20.02986	20.05812	1
ƒ4	1.32E+12	1.23E+12	1.07
ƒ5	18423850	15366069	1.2
ƒ6	1051798	1053990	1
ƒ7	6.2E+09	4.59E+09	1.35
ƒ8	8.09E+16	7.84E+16	1.03
ƒ9	1.29E+09	1.15E+09	1.12
ƒ10	94715529	94981824	1
ƒ11	9E+11	5.62E+11	1.6
ƒ12	7454.659	8437.836	0.88
ƒ13	2.61E+10	2.6E+10	1
ƒ14	6.27E+11	5.69E+11	1.1
ƒ15	6.61E+08	4.51E+08	1.46
t-test results	Win = 3	Win = 4	AVG=1.07
	Tie = 8		

Table 3.17: Effect of re-evaluating, NFC=60D

D=1000 NFC = 60D Number of runs=31			
Re-evaluating	NO	Yes	
Function	(a): 3-points CS	(b): 3-points CS	Ratio (a)/(b)
ƒ1	5.84E-08	5.84E-08	0
ƒ2	1846.641	1846.641	3.61
ƒ3	20	20	1
ƒ4	1.18E+12	1.4E+12	1.73
ƒ5	18144256	16673094	1.14
ƒ6	1045799	1047769	1
ƒ7	4.82E+09	5.52E+09	1.55
ƒ8	8.90E+16	6.36E+16	2.33
ƒ9	1.35E+09	1.16E+09	1.13
ƒ10	94132474	94323534	1
ƒ11	7.62E+11	6.17E+11	1.43
ƒ12	6470.146	6460.198	2.98
ƒ13	2.31E+10	2.37E+10	1.45
ƒ14	5.24E+11	5.86E+11	1.29
ƒ15	6.65E+08	4.65E+08	1.72
t-test results	Win = 3	Win = 3	AVG=1.56
	Tie= 9		

Tables 3.16 and 3.17 show the effect of re-evaluating in the first iteration. Re-evaluating shows some improvements. In the low-budget (NFC=20D) test, it improved around seven percent in average ratio. But for the medium budget, it shows 56 percent improvement in the average ratio. Re-evaluation needs extra FC, so in low-budget situations, it makes less progress.

3.6 Results Analysis

In this section, we try to find out which scheme works better under specific conditions. For this purpose, we perform the explanation into some major groups, which are low-budget, high-budget, and most potent algorithm to find the best solution.

3.6.1 Best Scheme for Low-budget

In Figure 3.11, the normalized results of the three best schemes are shown to have better insight into the difference of the results for each benchmark function. All of them are minimization problems, so the lower is better. Three-points CS with re-evaluation in the first iteration is the best one.

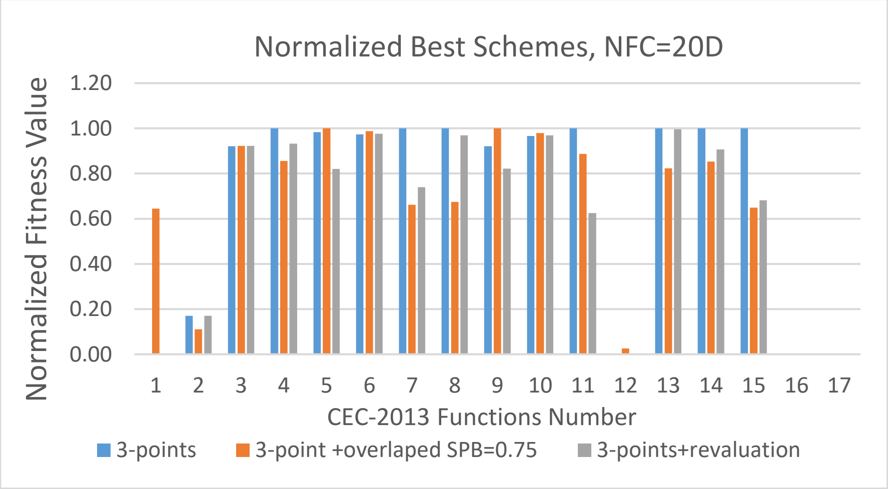


Figure 3.11: Comparing three best algorithms for low-budget (20D)

Based on the comparative results, which are reported in Figure 3.11, the three best algorithms all belong to the three-points CS scheme. So, we can say three-points CS works better when the algorithm finds reasonable results with low NFC. All three variations of the three-points CS have a slight difference in terms of the number of wins based on the t-test as well as the average on result ratio. But the three-point CS with re-evaluating for the first iteration is the best. Both three-points and overlapped three-points schemes are second best. Because the difference is tiny between winner schemes, for the sake of simplicity, we select three-point CS as the best scheme for the low-budget.

For better exploration, all schemes benefit from different permutation. We compered the results based on having different permutations in the first iteration of each run, or having different permutations in every iteration. Results show slight differences in favor of having different permutations in all iterations (Tables 3.13 and 3.14).

3.6.2 Best Scheme for High-budget

When it is possible to have higher NFC, the story totally changes, and three-points CS cannot perform relatively well. Instead, overlapped schemes perform significantly better. Among overlapped schemes, ones with SPB=0.9 are the best, which shows that less box-constraint cut in each iteration causes better results, but it needs more budget to converge.

In Figure 3.12, the normalized results of the three best schemes are shown to have better insight into the difference of the results for each benchmark function. The problem is minimization, so the lower is better.

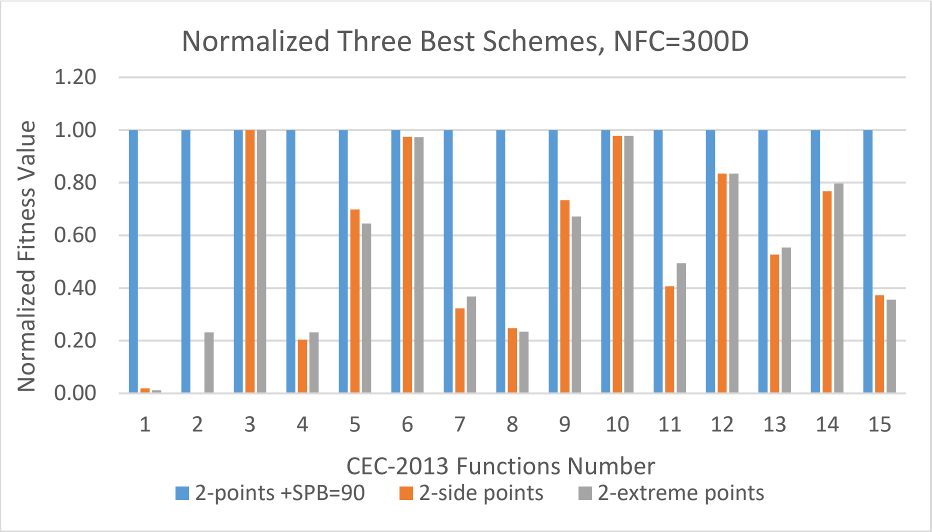


Figure 3.12: Comparing three best algorithms for high-budget (300D)

Among the three winner algorithms, which are two-points overlapped, two-side points, and two-extreme points, the two last ones show superior performance in all benchmark functions. These two algorithms benefit from this attitude where instead of selecting 90% of the box-constraint with one sample, they cut 10% of the box-constraint by one sample, which better represents that part. Two-side points and two-extreme points schemes are almost the same in terms of the average ratio, but in terms of t-tests, two-side points win three of 15 benchmarks, and two-extreme points win five of them. Moreover, the two-extreme points scheme is slightly simpler than the other one. So, the winner is the two-extreme points scheme for high-budget situations.

3.6.3 Best Scheme in Terms of Finding the Best Solution

In some situations, there is no limit for NFC or the optimizer needs less NFC than the limit, and the final result of the optimization is more important than the NFC, which is needed. In these situations, two-extreme points are the best scheme among all others based on the results reported above. It is noticeable that based on the no-free-lunch theorem, there is no algorithm to solve all problems, and our judgment to find the best scheme is based on the benchmark results. CEC-2013 is designed to contain a variety of problems with different complexities. So, we assume that if our scheme works well on CEC-2013, it is more likely to works well on many other applications. In the next section, we try to introduce more variations of this best scheme to make it suitable for more applications.

3.7 Proposing other Enhanced Variants of Two-extreme Points CS

After finding that the best scheme is two-extreme points in the previous sections, we can enhance the best scheme to have not only better results, but more generalized as well. In this section, we proposed different options that can be added to the main algorithm to make it either more accurate, or accelerated. We have tested all these options on the best scheme, which is two-extreme points CS. All of these options can be added to all previous schemes as well to make them more suitable for a specific application. These options are (1) expansion to find a better solution, (2) Stochastic Coordinate Search (SCS) to decrease the NFC, (3) Adaptive Coordinate Search Frequency (ACSF) to accelerate the convergence, (4) random initialization, and (5) population-based CS to have better exploration. In the following sub-sections, details are explained.

3.7.1 Expansion

In all different versions of the CS proposed in this work, some portion of the search space is eliminated in each iteration, and the optimizer can never search this portion of the search space in upcoming iterations. With some enhancements, CS can expand the search space in some defined conditions, and the optimizer has this chance to explore some of the regions which have been cut in previous iterations. Better exploration leads to finding better solutions, but the convergence rate suffers.

Assume that for variable one, CS cut the left side of the box-constraint of variable one, three times in a row (in three iterations in a row). Then a portion of box-constraint will be added to the right side of the box-constraint. This procedure is illustrated in Figure 3.13. This procedure helps the algorithm expand the box-constraints to find a better result. In the case of hard box-constraint, expansions beyond the initial box-constraint will be rejected. In this technique, two hyperparameters should be adjusted, which depend on the problem type and shape of the landscape. The first one is the Expansion Sequence (ES), which indicates after how many times cutting in the left (right) side in a row, expansion should be executed in the right (left) side. The second hyperparameter is the Expansion Amount (EA), which shows the amount of expansion. The downside of this technique is increasing the NFC, but because of the CS algorithm's fast convergence rate, this drawback can be handled. Figure 3.14 illustrates the performance plots of two-extreme points with and without expansion. These plots show that the expansion technique in some functions improves the results. In terms of convergence speed, it does not make noticeable improvements but in terms of final results in functions $f2,\ f4,\ f5,\ f8,\ f9,$ and $f15,$ it makes remarkable improvements. By adjusting the hyperparameters of the expansion technique as well as using it in an ensemble manner, we can take full advantage of this technique, as done in Chapter 4. Table 3.18 shows the effect of the expansion option on the two-extreme points CS algorithm. Results show a huge difference in six benchmark functions in favor of the two-extreme points CS with expansion.

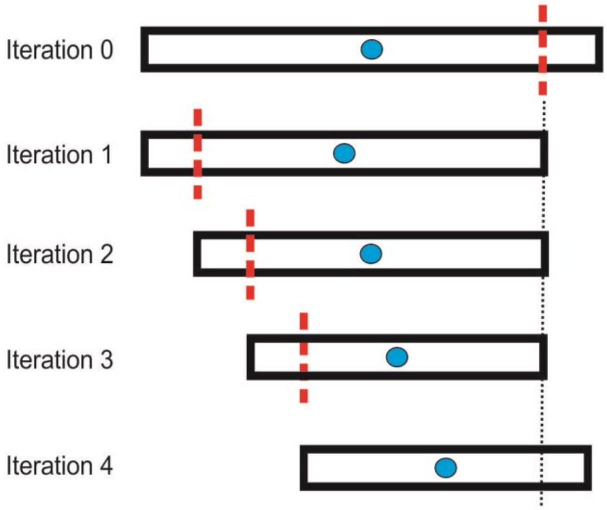


Figure 3.13: Expansion of box-constraint or search space

Table 3.18: Two-extreme points vs. two-extreme-expansion points CS, NFC=300D

D=1000 NFC = 300D Number of runs=31			
Function	**(a): 2-extreme points CS - SPB=0.90**	**(b): 2-extreme-expantion points CS - SPB=0.90**	**Ratio (a)/(b)**
ƒ1	3.95E-05	0.00734	0.01
ƒ2	28.5836	1.12E-07	2.55E+08
ƒ3	20.0033	20.0188	1
ƒ4	2E+11	9E+10	2.26
ƒ5	3777400	2621435	1.44
ƒ6	1000336	1000996	1
ƒ7	1.1E+09	1.6E+09	0.71
ƒ8	8.3E+15	4.2E+15	1.99
ƒ9	3E+08	2E+08	1.52
ƒ10	9.1E+07	9.1E+07	1
ƒ11	1.3E+11	1.4E+11	0.97
ƒ12	1044.28	987.871	1.06
ƒ13	8.3E+09	1.4E+10	0.61
ƒ14	3.4E+11	4.6E+11	0.74
ƒ15	2.1E+07	1448709	14.49
t-test results	**Win = 5**	**Win = 7**	**AVG=2.05** (ƒ2 excluded)
	Tie= 3		

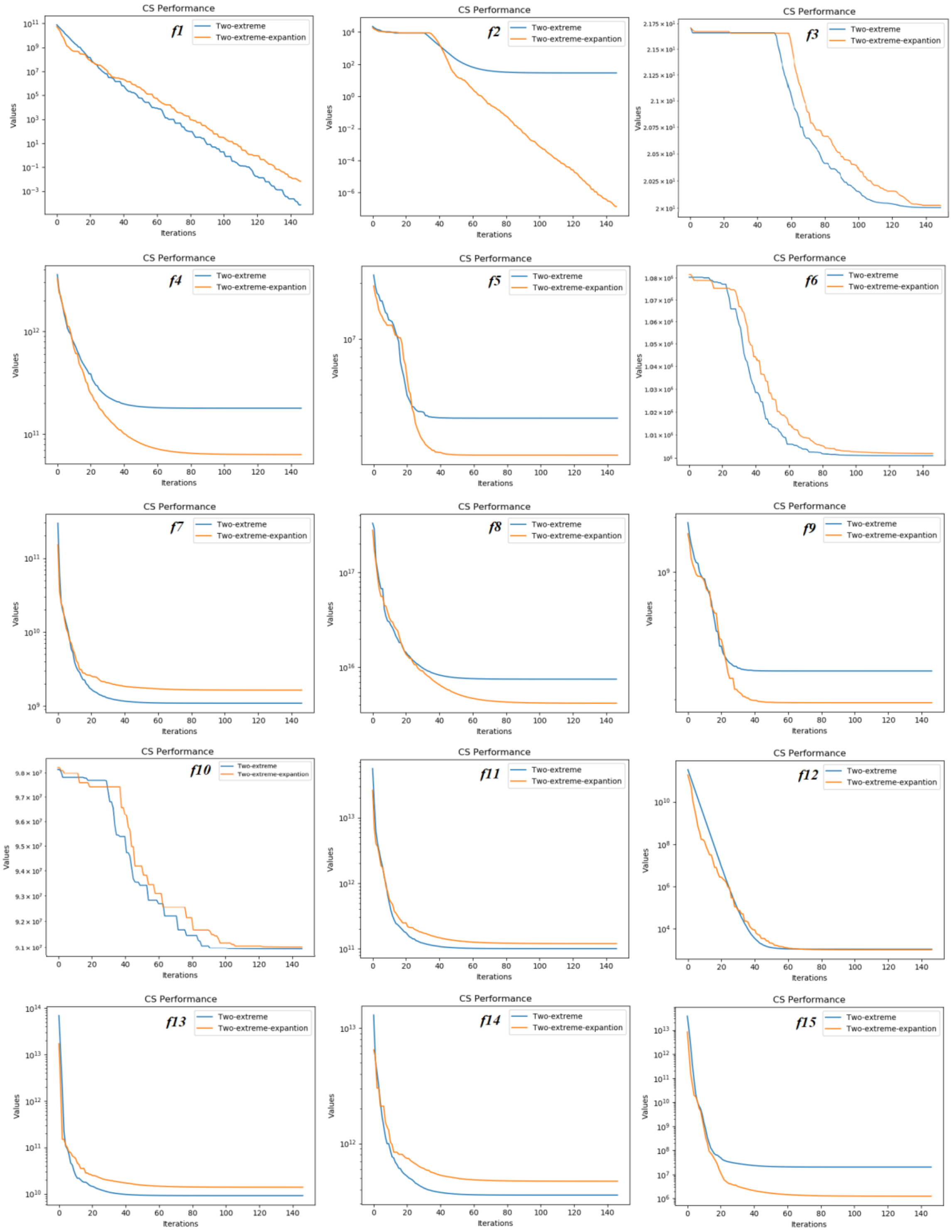


Figure 3.14: Performance plots for two-extreme points with or without expansion

3.7.2 Stochastic Coordinate Search (SCS)

For very large-scale problems like neural network optimization, SGD and SCD have been used, as mentioned in Chapter 2. We can use the same strategy for CS. In this technique, a selected portion of the variables is updated in each iteration, and other variables are untouched. Because this method is straight forward, we do not go through the details.

3.7.3 Adaptive Coordinate Search Frequency (ACSF)

Adaptive Coordinate Frequency (ACF) was proposed in [37] to accelerate CD. ACF does not treat all variables in the same way, and allocates more budget on selected variables, which are more important at a specific stage of optimization. Then, it adaptively updates the importance vector of variables in each iteration. ACF was proposed for convex problem optimization and has showed significant speed-up. We can use this technique with some modifications to speed-up the optimization process for CS. ACSF is similar to SCS, but the subset of variables will be selected by considering the sensitivity of the fitness function to each variable. For this purpose, the average of the fitness-value-changes for each variable is calculated in the first iteration, and then a probability vector for variables is constructed. Each variable that makes more fitness function changes will have more chances to be selected for the next iteration. This technique speeds up the optimizer because, on the landscape, it moves more on directions which are responsible for more changes in the fitness function. But in comparison to SCS and Full-CS (which treats all variables the same), the final result is worse because ACSF does not treat all variables the same. Figure 3.15 illustrates the performance plots for three schemes, which are two-extreme points CS, SCS, and ACSF. Table 3.19 reported the results for these three schemes.

Table 3.19: Comparing with two-extreme points, SCS, and ACSF, NFC=300D

D=1000 NFC = 300D Number of runs=31			
Function	**2-extreme points CS - SPB=0.90**	**SCS**	**ACSF**
ƒ1	3.95E-05	0.00134	4.45E-06
ƒ2	28.5836	28.5864	28.581
ƒ3	20.0033	20.0045	20.0032
ƒ4	2E+11	3E+11	5.8E+11
ƒ5	3777400	4063274	4494813
ƒ6	1000336	1005014	1026041
ƒ7	1.1E+09	2E+09	2.2E+09
ƒ8	8.3E+15	1.20E+16	1.71E+16
ƒ9	3E+08	2.9E+08	3.6E+08
ƒ10	9.1E+07	9.2E+07	9.4E+07
ƒ11	1.3E+11	1.9E+11	2.8E+11
ƒ12	1044.28	13214.2	1083.71
ƒ13	8.3E+09	1.3E+10	1.4E+10
ƒ14	3.4E+11	4.7E+11	4E+11
ƒ15	2.1E+07	7.1E+07	3.4E+07

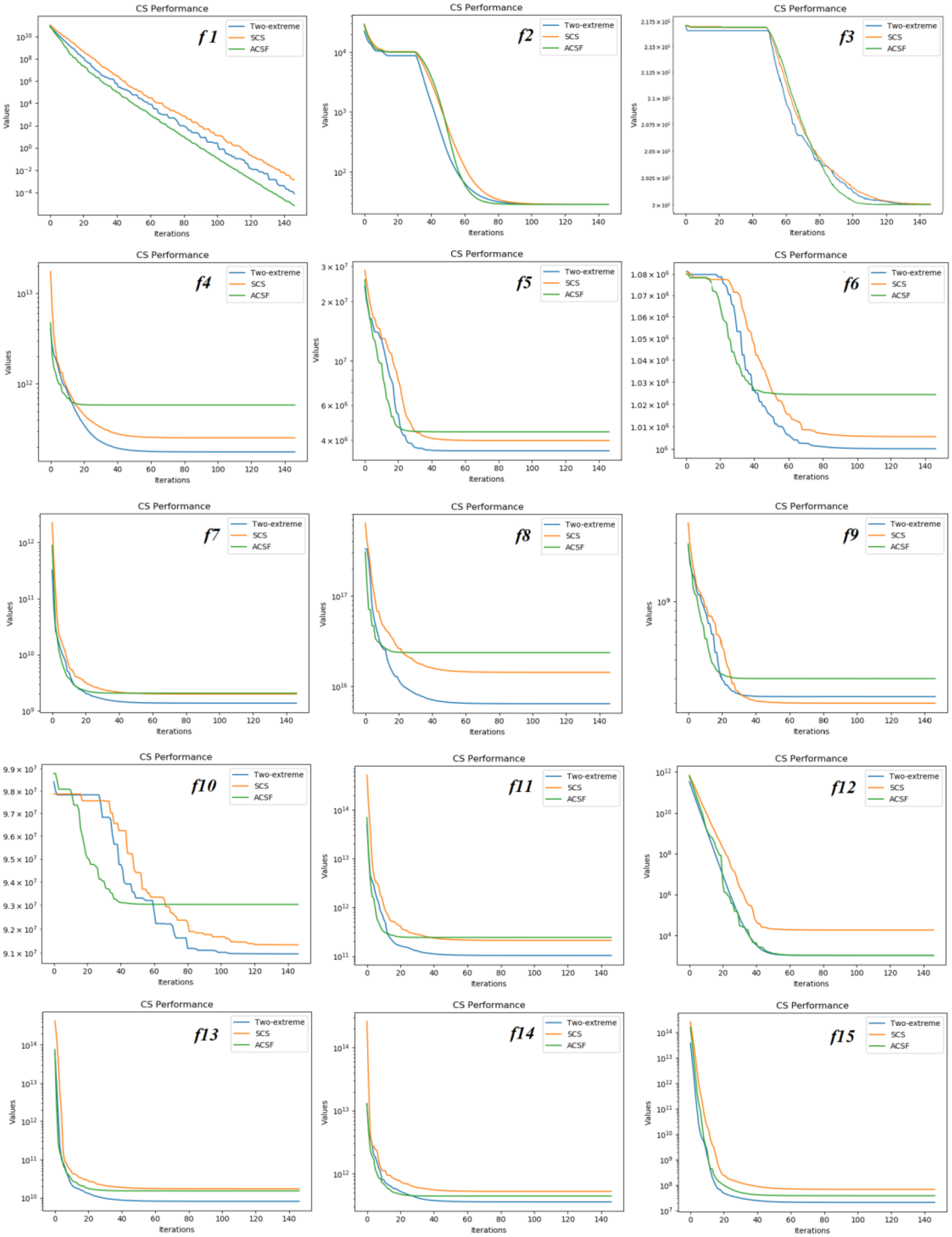


Figure 3.15: Performance plots for two-extreme points, SCS and ACSF

Results show, ACSF has a better convergence rate in early stages, as we expected, which shows the speed-up property of the ACSF. ACSF moves more in directions that lead to more improvement. SCS has worth convergence rate because it changes some of the variables in each iteration. In terms of final results, the best scheme is two-extreme points, because it considers all variables. The second-best is SCS, and the third one is ACSF.

3.7.4 Random Initialization

In CD methods, one variable is selected at a time while other variables are set to the center of their box-constraints. For example, if the box-constraint for all variables is [-1,1], then the center points are zero. In some cases, such as fully connected neural networks, if we set all weights (except one weight) to zero, the output will be zero. So, it shows that in some situations, random start points will be beneficial to help the optimizer make progress in finding the solution. Otherwise, the optimizers may face a flat landscape, and in flat landscapes, the optimizer cannot make the right decision. Moreover, random initialization helps the optimizer start from different points on the landscape, which result in better exploration, and increases the chance of finding the global solution.

Random initialization is not straight forward for CS. If we randomly select initialization points, all variables will be at the center of their remained box-constraint after one iteration. Therefore, this random initialization will not have much effect on the optimizer progress. Also, in CS, when we want to change one variable, instead of setting other variables to the center of their box-constraint, we can randomly select them. But the algorithm will not converge to the solution, because we look at totally different points on the landscape in each step. The points are not close to each other, and there is high diversity. So, the optimizer cannot find a solution among very diverse testing points.

To address random initialization for the two-extreme points scheme, we came up with two different methods. The first one is random initialization of the start points, which then constructs the box-constraints around them for each variable, meanings that the random start points are in the center of the box-constraint. In this approach, each box-constraint is constructed by the size of the box-constraint, and the center (start) point. So, we cannot have a pre-defined value for each box-constraint boundary, which can be problematic when the problem has hard box-constraint. This means that boundaries of each box-constraint are important and cannot be violated, CEC-2013 benchmarks for instance.

The second method is randomly initializing points for each variable in the box-constraint range. In each iteration, when the algorithm cuts 10% of the left side of the box-constraint for example, we shift the random initial point to the right, and if it cut 10% of the right side of the box-constraint, we shift the random initial point to the left. The amount of this shift is proportional to the cut portion. Eq. (3.9) and Eq. (3.10) show how the amount of this shift is obtained.

$$RS = C_P \times (P_V - U) \tag{3.9}$$

$$LS = C_P \times (L - P_V) \tag{3.10}$$

In this equation, RS is the amount of the right shift, and LS represents the amount of the left shift. C_P is the cutting percentage. U stands for upper bund, and L stands for lower bund of the box-constraint. P_V is the value of the random initial point. In each iteration P_V updates. Figure 3.16 shows how this method works. In this picture, the blue dot represents P_V.

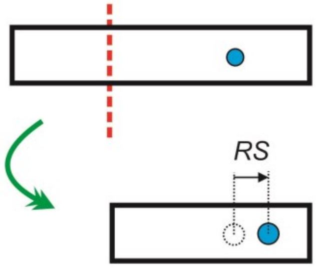


Figure 3.16: Updating Pv in random initialization when box-constraint is cut on the left side

The second method for random initialization does not have the limitation of the first method, meaning that the second method can be applied for all problems with any kind of box-constraints. So, the second method can be considered a universal method for random initialization for CS and CD algorithms. If all initialization points are selected in the center of the box-constraint, it will be just like the basic method of initialization. Therefore, this method of random initialization is a generalized technique for different methods of initialization. We use this method to randomly initialize weights (normal random μ=0 and σ=0.1) for neural network training in Chapter 5. But in CEC-2013 benchmarks, center point initialization shows better results. So, we use center points for the benchmark problems.

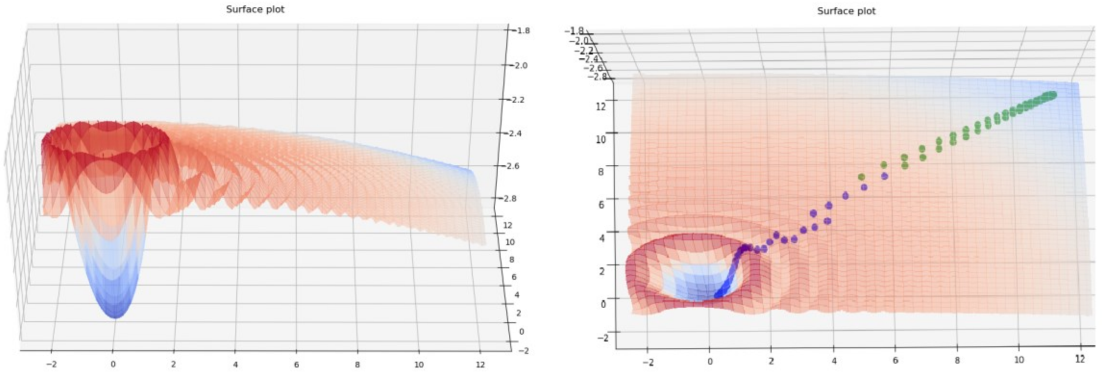
In the literature review section, we mentioned that center-based initialization could have positive effects on the performance of the optimization. Moreover, random initialization has its own benefits, as we have mentioned previously. Therefore, we want to take advantage of the center and random initialization at the same time. To address this, we can use normal random initialization, and set the mean value of normal distribution to the center of the box-constraints. Standard deviation can be close to zero to ensure all variables are initialized close to the center point.

We have tested this method on CEC-2013, but results were not promising. On the contrary, random initialization for neural network weight optimization outperformed the original initialization method. We reported the results on neural network weight optimization in Chapter 5.

3.7.5 Population-Based CS

All previous proposed CS algorithms can be considered as single-solution methods, which has one point in the search space moves towards the solution. For better exploration, a population-based technique can be proposed. Moreover, the population-based technique paves the way to benefit from parallelization. In population-based approaches in each iteration, instead of one vector of the solution, some vectors of solutions are generated, and used for better decision making to move towards the global solution. Population-based algorithms work very well for low dimensional problems, and find global solutions by supporting exploration capability [53]. Moreover, these methods have more robust results, especially in the presence of noise [54]. Figure 3.17 shows the pass towards a solution for the single and population-based algorithm on a deceptive landscape. The population-based one can find the global solution, while the other one is deceived. In one of our works [55] population-based algorithm based on coordinate search has been used. In this paper, the number of individuals in the population is 50, and in each iteration, 50 different permutations are generated to make the initial population. Each of the permutations are evaluated, and based on fitness value, 20 percent of the best ones are selected. After that, by getting a vote on selected permutations, a decision is made to select the left, right, or center of box-constraint. To be fair for comparing the results with the single-solution scheme, the single-solution algorithm runs 50 times separately with different permutations for each run, but there is no voting or collaboration between the solutions. Finally, we keep the best result among the 50 separate runs.

Results on 30 functions of CEC-2017 benchmarks show the superior performance of the population-based algorithm for dimension 50 (15 wins and three losses) and dimension 100 (17 wins and four losses) in front of a single-solution optimizer. But for dimension 1000, population-based method does not show any improvements and results are almost the same. By increasing the dimension, search space volume will increase exponentially, but population size cannot increase exponentially, so the exploration ability of this method declines. This shows that for large-scale problems, it is better to stay with single-solution-based CS algorithms.



(a) Landscape

(b) single-solution (green) vs. population-based (blue)

Figure 3.17: Deceptive landscape (a), single-solution (green dots) and population-based (blue dots) two-extreme points CS (b)

3.8 Comparative infographics of Proposed Schemes

At the end of this chapter, to summarize the results, two comparative infographics are used in Figures 3.18 and 3.19 to show the difference between the schemes for different levels of budget. The first, second, and third best schemes for each group are shown in the figures.

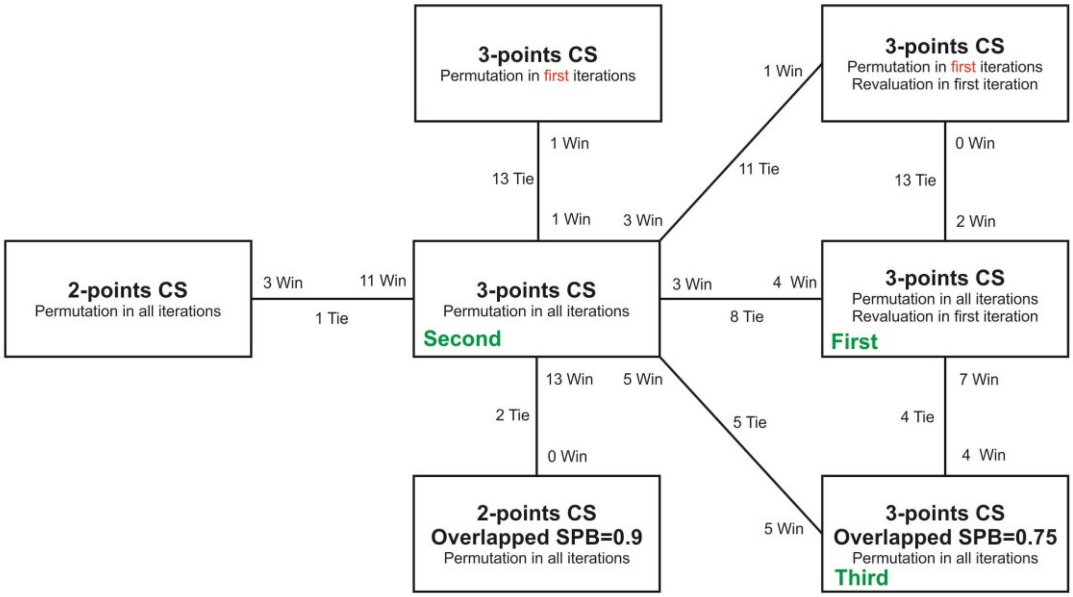

3-points CS
Permutation in first iterations
3-points CS
Permutation in first iterations
Revaluation in first iteration
1 Win
1 Win
13 Tie
1 Win
3 Win
11 Tie
0 Win
13 Tie
2 Win
2-points CS
Permutation in all iterations
3 Win
11 Win
1 Tie
3-points CS
Permutation in all iterations
Second
3 Win
4 Win
8 Tie
3-points CS
Permutation in all iterations
Revaluation in first iteration
First
13 Win
5 Win
2 Tie
5 Tie
0 Win
7 Win
4 Tie
4 Win
2-points CS
Overlapped SPB=0.9
Permutation in all iterations
5 Win
3-points CS
Overlapped SPB=0.75
Permutation in all iterations
Third

Figure 3.18: Comparative infographics of schemes for low budget, NFC=20D

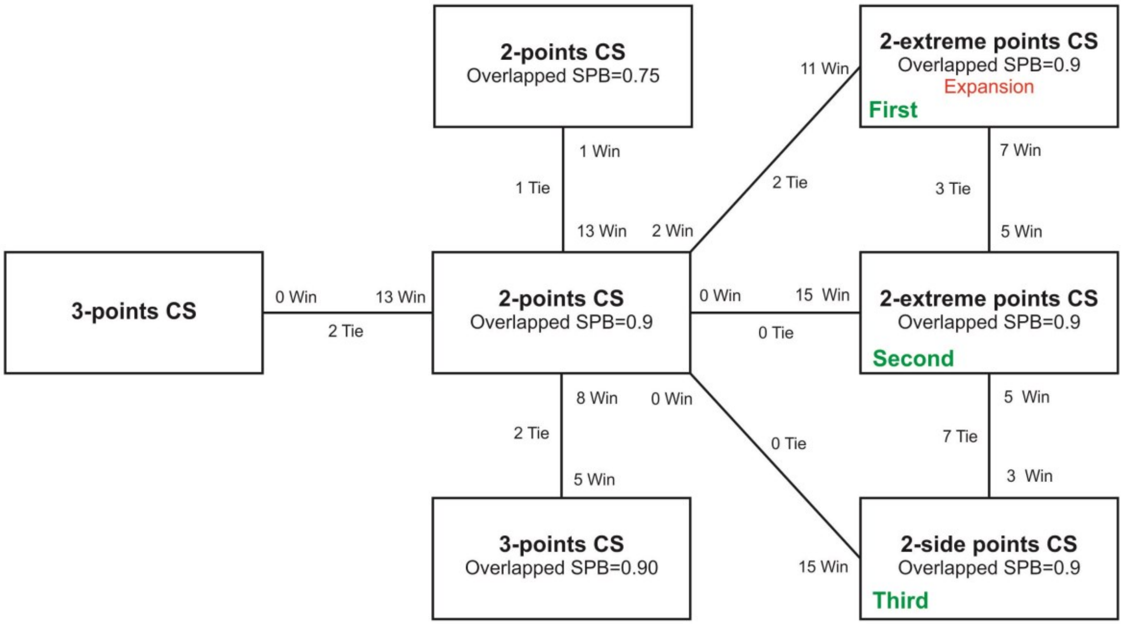


Figure 3.19: Comparative infographics of schemes for high budget, NFC=300D

3.9 Conclusion

In this chapter, we developed a gradient-free search algorithm for non-convex LSGO. The best scheme for a low-budget is three-points CS, and the best scheme for a high-budget is two-extreme points. NFC is a limiting factor in the optimization procedure. These algorithms can optimize large-scale problems with relatively low NFC because ccomplexity of the CS algorithm is O(D), it means linear complexity over the dimension, which is really low complexity

Now that the characteristics of each algorithm are known, and based on the application, one of these algorithms can be used for optimization purposes. Based on this knowledge, in the following chapters, two-extreme points CS will be tested in different domains to measure the performance of the algorithm in different situations.

Chapter 4

Comparing Proposed Method with State-of-the-art Algorithms

In order to evaluate the best proposed algorithm, we tested it on a well-known large-scale benchmark. From 2013 until now, the CEC-2013 benchmarks are used to measure the performance of large-scale single objective optimizers. Based on the results reported in [56], we selected the four best algorithms with higher ranks to evaluate the performance of our proposed optimizer in comparison with other state-of-the-art algorithms. Table 4.1 shows these methods, ranks, and the year that they proposed them.

Table 4.4.1: State-of-the-art algorithms for CEC-2013

Algorithm	Rank	Proposed year
MLSHADE-SPA [56] Memetic framework with Linear population size reduction for Success History-based Differential Evolution and Semi-Parameter Adaptation	1	2019
MOS2013 [57] Multiple Offspring Sampling	2	2013
VGDE [58] Variable Grouping based Differential Evolution algorithm	3	2014
IHDELS [59] Iterative Hybridization of DE with Local Search	4	2015

Budget in this competition is 3,000,000 function evaluation (FE). This number of function evaluations (NFE) is very high for our algorithm because our scheme can converge to the solution with less than 200,000 FE on average. For using the extra budget, we have two strategies. The first one is re-running the algorithm with different permutations, and the second one is using different hyperparameters in each run. The second technique can be done by using an ensemble model with two different variations on our best algorithm. These strategies are explained in detail in the following parts. Based on our comparison, the best scheme of the CS method for the CEC-2013 is 2-extreme points with expansion. But we use some enhancements to elevate the performance of the algorithm (Algorithm 4). These enhancements are as follow:

a- The budget is 3,000,000 FE. It is very important to use this budget in an effective way. Because 2-extreme CS needs much less NFE to converge, we re-run the algorithm with different permutations. In each permutation, the order of variables randomly changes, leadings to a different result. The best result is preserved, and considered as the algorithm results for each run.

b- CS needs different NFE for each function in the CEC-2013 benchmark to converge to the solution, and it also depends on different things such as permutation. If we randomly change the permutation during the process, the exact number of the FE cannot be estimated. Therefore, instead of predefined NFE, we calculate the change amount (not just improvement) for each iteration. If the change is less than $0.001 \times f_c(x)$ for 20 iterations in a row, the algorithm will re-run. $f_c(x)$ represent the current fitness value of the function, and x stands for the variables. By using this technique, we make sure that the extra budget is not wasted, and for each run, we just use the NFE that is needed. The 0.001 is multiplied by the $f_c(x)$, and the number 20 for the number of iterations in a row is selected by trial and error experiments.

c- We realized that the two-extreme CS with expansion has some important hyperparameter, which has a noticeable effect on the performance of the algorithm. The most important one, based on our empirical results, is the expansion sequence (ES). ES defines how the expansion in box-constraint is applied. For example, if the ES equals to two, after two left cuts in a row or two right cuts in a row, box-constraint is expanded in the opposite direction. Our experiment shows that selecting ES equal to two or four, result in better performance. But the problem is that in some functions two is better, and in other ones, four is the best. So, we decided to take advantage of both of them in an ensemble way. Different reruns are applicable for the CS, as we mentioned above, and for the first run, ES is equal to four, and for the next run, ES is equal to two, and is alternatively changed in this way until the last run. The best result is preserved and represents the result of the algorithm after 3,000,000 FE. The permutation changes randomly in each iteration, so the result will not be deterministic. The average of the best results over 25 runs is calculated and reported as the algorithm results.

Algorithm 4 The pseudo-code of two- extreme points CS with expansion

Input: $\varkappa \epsilon \mathbb{R}^D$: Solution vector, D: Dimension of the problem, U: upper bond, L: Lower bound, N_{ite}: Number of Iterations, BSF: Box-constraint shrinkage factor, ES: expansion sequence

Output: S: The best solution found so far

```
$C \leftarrow 0$
$ES \leftarrow 1$
for $j \leftarrow 1$ to $N_{ite}$ do
    if $NFC > C$ then
        Brake
    $C_i \leftarrow \frac{U_i - L_i}{2}$
    for $i \leftarrow 1$ to $D$ do
        $LS_i \leftarrow L_i$
        $RS_i \leftarrow U_i$
        $S_{i_L} \leftarrow f([C_1, C_2, ..., LS_i, ..., C_D])$
        $S_{i_R} \leftarrow f([C_1, C_2, ..., RS_i, ..., C_D])$
        $C \leftarrow C+2$
        if $S_{i_L} < S_{i_R}$ then
            $U_i \leftarrow U_i - (U_i - L_i) \times (1 - BSF)$
            $C_i \leftarrow \frac{U_i - L_i}{2}$
            $S \leftarrow S_{i_L}$
        end
        if $S_{i_L} > S_{i_R}$ then
            $L_i \leftarrow L_i + (U_i - L_i) \times (1 - BSF)$
            $C_i \leftarrow \frac{U_i - L_i}{2}$
            $S \leftarrow S_{i_R}$
        end
    end
end
if $\Delta S < 0.001 \times S$ for 20 iterations in a row then
    $ES \leftarrow$ Toggle between 1 and 3
    Restart
end
```

Table 4.2 shows the mean value of the results over 25 runs. In five benchmark functions, the proposed scheme has better results than the other ones. Moreover, for functions one and two, CS can reach the global solution (zero). The second best is MLSHADE-SPA that wins in four benchmarks, but cannot reach the global ones. The standard deviation of the results over 25 runs can be seen in Table 4.3. This is remarkable that the proposed CS algorithm has a relatively lower standard deviation in nine functions, and it shows the robustness of the solution.

Table 4.4.2: Mean value of the results

Function	Proposed CS	MLSHADE-SPA	MOS2013	VGDE	IHDELS
f1	**0.00E+00**	1.94E−22	**0.00E+00**	**0.00E+00**	4.34E−28
f2	**0.00E+00**	7.89E+01	8.32E+02	4.56E+01	1.32E+03
f3	2.15E+01	**9.96E−14**	9.17E−13	3.98E−13	2.01E+01
f4	1.55E+10	6.90E+08	**1.74E+08**	5.96E+08	3.04E+08
f5	**1.62E+06**	**1.80E+06**	6.94E+06	3.00E+06	9.59E+06
f6	9.97E+05	**1.40E+03**	1.48E+05	1.31E+05	1.03E+06
f7	9.05E+08	5.31E+04	1.62E+04	**1.85E+03**	3.46E+04
f8	7.98E+14	9.77E+12	8.00E+12	7.00E+14	**1.36E+12**
f9	**1.08E+08**	1.61E+08	3.83E+08	2.31E+08	6.74E+08
f10	9.07E+07	6.56E+02	9.02E+05	**1.57E+02**	9.16E+07
f11	2.85E+10	4.04E+07	5.22E+07	7.52E+07	**1.07E+07**
f12	9.87E+02	**1.04E+02**	2.47E+02	2.52E+03	3.77E+02
f13	7.36E+09	7.21E+07	**3.40E+06**	1.36E+09	3.80E+06
f14	3.27E+11	**1.52E+07**	2.56E+07	2.29E+10	1.58E+07
f15	**1.22E+05**	2.76E+07	2.35E+06	3.44E+06	2.81E+06

Table 4.4.3: Standard deviations of the results

Function	Proposed CS	MLSHADE-SPA	MOS2013	VGDE	IHDELS
f1	0.00E+00	4.79E−22	0.00E+00	0.00E+00	1.23E-27
f2	0.00E+00	9.69E+00	4.48E+01	3.25E+01	6.98E+01
f3	0.00E+00	7.91E−15	5.12E-14	1.42E-14	1.36E-01
f4	4.21E+09	4.41E+08	7.87E+07	4.45E+08	1.07E+08
f5	2.13E+05	2.34E+05	8.85E+05	5.29E+05	2.03E+06
f6	4.34E+02	2.39E+03	6.43E+04	1.74E+04	1.95E+04
f7	2.54E+08	1.96E+04	9.10E+03	3.39E+03	1.33E+04
f8	2.8E+14	5.53E+12	3.07E+12	3.29E+14	6.85E+11
f9	1.31E+07	1.94E+07	6.29E+07	4.01E+07	1.30E+08
f10	6.56E+04	2.40E+02	5.07E+05	2.51E+01	9.17E+05
f11	8.39E+09	1.98E+07	2.05E+07	2.16E+07	4.12E+06
f12	1.65E-01	7.64E+01	2.54E+02	2.81E+02	3.30E+02
f13	1.28E+09	4.99E+07	1.06E+06	7.01E+08	9.72E+05
f14	2.76E+10	3.08E+06	7.94E+06	1.91E+10	5.11E+06
f15	2.16E+4	9.01E+06	1.94E+05	2.43E+05	1.01E+06

Chapter 5

Neural Network Training with CS

In order to evaluate the proposed algorithm on another very large-scale optimization problem, we selected neural network weight optimization as a benchmark. Weight optimization is called training, and it can be very challenging because in neural networks, there are thousands or even millions of weights, so the dimension of the optimization problem can be huge. High-dimensional problems are very expensive in terms of time and computational cost especially, for non-gradient based algorithms. Metaheuristic or swarm algorithms such as DE or PSO can be used for training, but they are very time consuming, or used for a very small size network with a small number of weights [60],[61],[62],[63]. Therefore, almost in all cases, gradient-based training methods are preferred. In this research, we show that the proposed CS algorithm can be used for training with a personal computer by just one CPU in a reasonable time, which is not feasible for other gradient-free algorithms.

5.1 Fully Connected Neural Network Training by Proposed CS

Almost in all structures of neural networks, one or more fully connected layers are used. So, for the case study, we selected a fully connected one with two hidden layers [64] as a framework. The first one has 300 nodes, and the second one has 100 nodes. For handwriting digit recognition, digits come in 28×28 pixels images. Each image has been flattened to a 1D vector by assigning each pixel value to each vector parameter value. Therefore, we have a vector with 28×28 =784 parameters. So, the input size of the network was chosen equal to the input vector dimension, which is 784. The output layer has ten nodes, which represent ten classes, and each class is linked to one digit. Data set for training and testing is MNIST. MNIST has 60,000 images for training and 10,000 images for testing. Figure 5.1 illustrates the network structure and the number of nodes for each layer. Activation functions for hidden layers one and two are ReLU, and for the output layer is Softmax. Loss-function is Categorical Cross-Entropy.

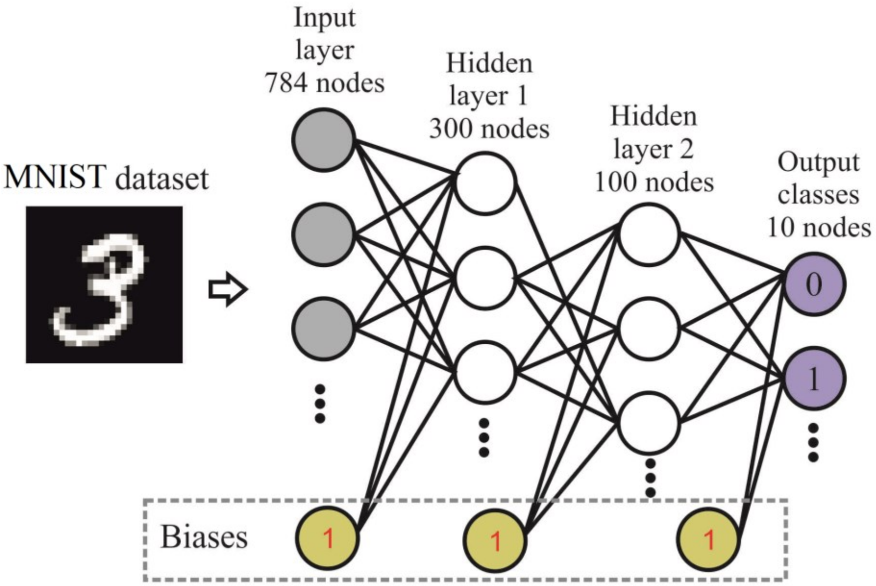


Figure 5.1: Structure and number of nodes in the fully connected network

Eq. (5.1) to (5.4) show the number of weights in this network. In these equations, weights between layers one and two for instance, can be calculated by multiplying the number of nodes in layers one and two. Moreover, for each node in each layer, there is a bias which is considered a weight. So, the number of biases should be added to the number of weights. The input layer is exempt and has no biases for nodes.

$$W_{Layer\ 1-2} = (N_{Input\ layer} + 1) \times N_{Hiden\ layer\ 1} = 785 \times 300 = 235{,}500 \tag{5.1}$$

$$W_{Layer\ 2-3} = (N_{Hiden\ layer\ 1} + 1) \times N_{Hiden\ layer\ 2} = 301 \times 100 = 30{,}100 \quad (5.2)$$

$$W_{Layer\ 3-4} = (N_{Hiden\ layer\ 2} + 1) \times N_{Output\ layer} = 101 \times 10 = 1{,}010 \quad (5.3)$$

$$W_{Total} = 235{,}500 + 30{,}100 + 1{,}010 = 266{,}610 \quad (5.4)$$

In this network, we face a very high-dimensional optimization problem. Gradient-based algorithms can train the network with high accuracy. To the best of our knowledge, there is no feasible alternative that can train such a network on a personal computer with just one CPU besides the proposed method.

For assessing the performance of the training methods, we need some metrics. Accuracy can be used as a metric, but it has a major flaw for classification problems. For example, we have a classification problem with two classes. If the model prediction returns only one class, the accuracy of the model is 50%, which shows a hidden issue. This shows that the model is unable to distinguish another class. Therefore, it is better to also use other metrics in order to cover all key factors for better measurement of the model performance in terms of distinguishing the right class. These metrics are precision and recall. Precision shows how many of the predicted members are correct in that particular class, and recall shows how many members of a particular class are retrieved correctly. Before explaining the equations of precision and recall, four basic elements should be defined. Assume we have two classes, and there is a model that predicts these two classes. The prediction can be labelled as positive or negative for the classes. Moreover, the prediction can be true or false. For instance, if the model predicts the positive class as positive, it is true positive. But, if it predicts the positive class as negative, it is a false negative. Eq. (5.5) to (5.7) show how precision, recall, and accuracy can be calculated. In these equations, T_P stands for true positive, T_N stands for True negative, F_P stands for false positive, F_N stands for false negative. For classification problems with more than two classes, it can be broken into some binary classification subsets, and then treated as explained above.

$$Percision = \frac{T_P}{T_P+F_P} \tag{5.5}$$

$$Recall = \frac{T_P}{T_P+F_N} \tag{5.6}$$

$$Accuracy = \frac{T_P+T_N}{T_P+T_N+F_P+F_N} = \frac{T_{Total}}{Total\ members} \tag{5.7}$$

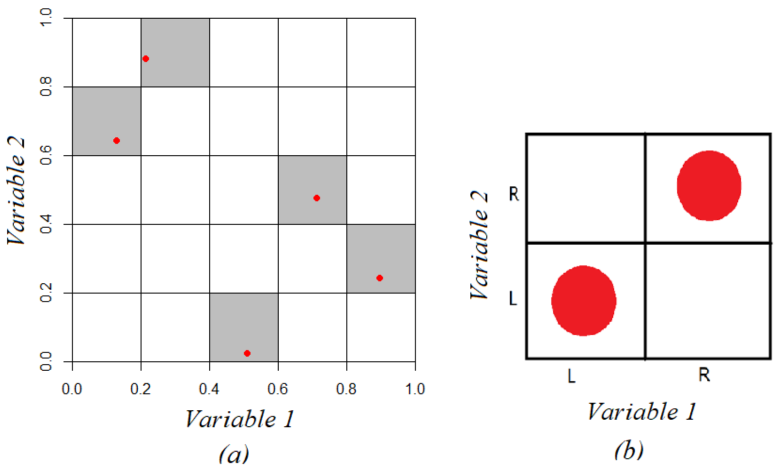
For this very high dimension problem, non-gradient based algorithms such as DE can be very time consuming, and this makes them impractical for training such a network. On the contrary, because the CS algorithm shrinks the search space exponentially, there is hope that a network with thousands or even millions of weights can be trained by it. We have tried to tailor CS for training fully connected networks. For this purpose, the best schemes of the CS have been selected, which is two-extreme points scheme. We have performed many tests on this network by different schemes and hyperparameters of CS algorithms, and tried to find the best combination of parameters for selected CS schemes.

The three-points CS scheme obtains less accuracy compared to the two-extreme point scheme. This result shows consistency with CEC-2013 benchmark results. In both of them, the two-extreme points method performs better, but it is more demanding in terms of time and computational power. BSF in the two-extreme points technique is far less than the three-points one. Therefore, it converges to the solution gradually. For this neural network training, which needs to adjust 266,600 parameters, these methods are time-consuming.

Therefore, we tried to change them to work faster by sacrificing a small amount of accuracy, but it makes the optimization method feasible for training such a network. For the accelerating train procedure, we can apply three methods; the first one is SCS, the second one is ACSF, and the third one is Bundling weights which we proposed is based on the Latin Hypercube Sampling (LHS) technique.

5.2 Latin Hypercube Sampling (LHS)

Assume that there are two variables in an optimization problem, and some samples need to be taken. Samples should be taken from the places in the search space, which are farther from each other as well as covers more of the search space. So, the search space can be divided into several same-size squares (hypercubes), and then samples are taken from each one. For high-dimension problems, the number of hypercubes grows exponentially, and sampling from each of them is impossible unless the size of hypercubes is very large. Then it is not beneficial to take one sample from a very large hypercube. By LHS, instead of sampling from each hypercube, samples are taken randomly in a way that all intervals are covered. Therefore, the number of samples is equal to the number of one variable interval. This method shows superior performance by a lower number of samples [65],[66]. For CS, if we want to change two variables at the same time, there are two options for each one, left or right. So, space can be divided into four quarters. By selecting two of them, all intervals can be covered, and for CS, we pick diagonal ones. Figure 5.2 (a) shows how LHS works, and part (b) illustrates how LHS can be used for CS. R stands for sampling from the right side, and L represents the sampling form the left side.



(a)

(b)

Figure 5.2: (a) Sampling based on the LHS, (b) Using LHS for CS

5.3 Bundling Variables Based on LHS for CS

To speed-up the training procedure, variables can be bundled (grouped) based on LHS. For applying this method, all weights are bundled in n groups where, $n = N_w/BS$, N_w is the number of variables (weights), and *BS* stands for bundle size. Then all weights in each group are considered simultaneously as one variable/weight. Then the CS selects an extreme right or an extreme left point in their box-constraint. For the next iteration, variables are shuffled, and then weights are bundled again. Figure 5.3 shows the procedure of bundling, and there are nine weights (W1 to W9) that are grouped into three bundles. Each weight would be grouped with different weights in each iteration, so it has the chance to eventually show its effect on fitness value.

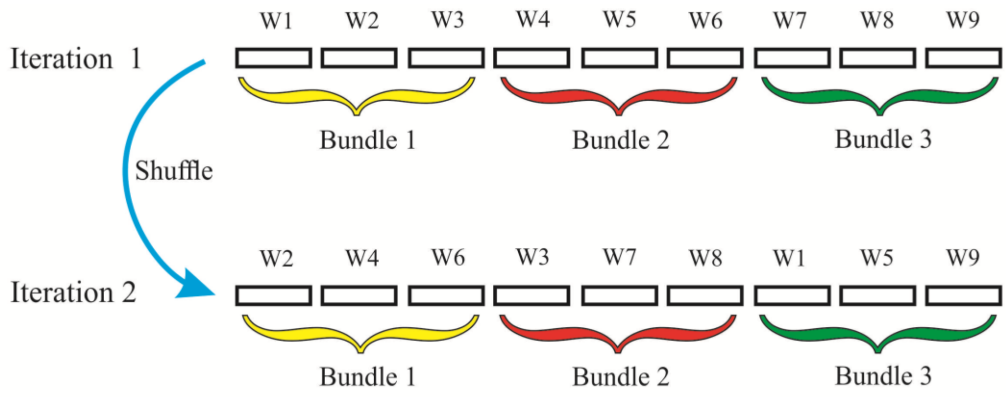


Figure 5.3: Bundling procedure

For comparing the results with other speed-up methods, we trained the network with SCS and ACSF. Results show that bundling weights result in better accuracy on the test dataset (Table 5.1). Accuracies are reported in percentages.

Table 5.1: Speed-up methods for training FCNN

	Bundling	**SCS**	**ACSF**
Train Accuracy	**94.73**	92.10	92.81
Test Accuracy	**93.82**	90.23	91.52

5.4 Effect of Different Initialization

For training a neural network, initialization is a very important phase. There are different approaches for this purpose. The first one is center initialization which is a wise choice for many applications, and more importantly for high dimensional problems [51]. But in this special case, the center point of the box-constraint is zero, and if all variables (except one bundle) are set to zero for initialization, CS can hardly start to move towards the optimum point. By changing one bundle while other bundles are zero, it is more likely to face multiplication by zero, which results in zero for the output of the network. To prevent this problem, we can choose a large number for bundle sizes to change many weights simultaneously. Then inputs can find a non-zero pass to the output of the network, but as we will explain in the sub-section 5.6, accuracy suffers by choosing large bundle size.

Another way to initialize weight is by random initialization. This random initialization can be uniform or normal. By normal distribution, when μ=0, we can benefit from center point initialization as well as avoid a lot of multiplication by zero. Table 5.2 shows the accuracy (in percentage) of the network by different initialization approaches. Random normal initialization (μ=0 σ=0.1) results in better accuracy.

Table 5.2: Effects of different initialization approaches on test accuracy

Fixed initialization	**Random initialization**					
Center	**Uniform**		**Center normal distribution**			
Value = 0	**_R_ ϵ [-2,2]**	**_R_ ϵ [-1,1]**	**μ=0 σ=0.05**	**μ=0 σ=0.1**	**μ=0 σ=0.2**	**μ=0 σ=0.3**
93.71	91.2	92.63	93.64	**93.82**	92.52	92.3

5.5 Different Data Feeding Method

Feeding whole train data for optimization improves the accuracy, but in terms of training efficiency, it is not the best choice. Instead of using the whole data for training, data can be separated into different folds, and for each fitness call (to calculate the error value), a fold can be used for training. It is very similar to batch training [67]. Whereas different folds are used, the network will see all the training data in different phases, and we can expect that accuracy does not suffer very much. Feeding a fold at a time helps the network trains faster. For example, if the training data is separated into six-folds, the training phase can be accelerated by a factor of six. Feeding different folds can be done in two ways. The first one is feeding separate folds, and the second one is starting with a fold, and sliding the fold over training data. Figure 5.4 illustrates three different ways of feeding data. Feeding separate folds causes more fluctuations in training accuracy (Figure 5.5 (b)). But feeding sliding folds causes fluctuations with less frequency. However, as shown in Figure 5.5 (c), when it slides over training data, it reaches to the end of the trading data and wants to start to form the first of the dataset, a spike (shown by arrows in Figure 5.5 (c)) will appear in the performance plot. The sliding amount in this test is 100. Table 5.3 shows the effect of different data feeding on the accuracy. The best accuracy can be reached by feeding aggregate data for each FC, but the fastest speed can be obtained by separate or sliding folds.

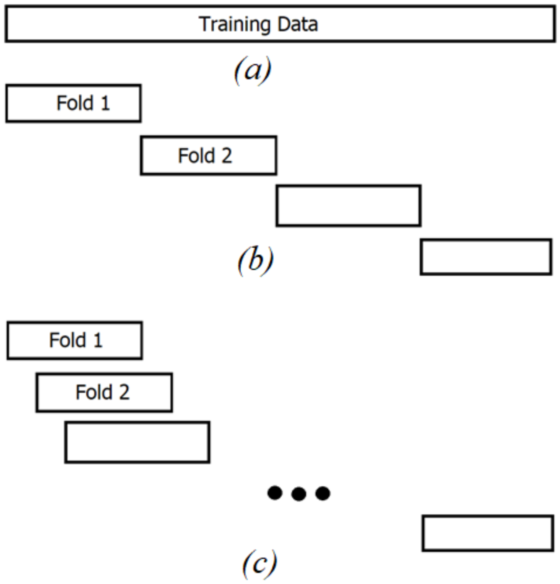
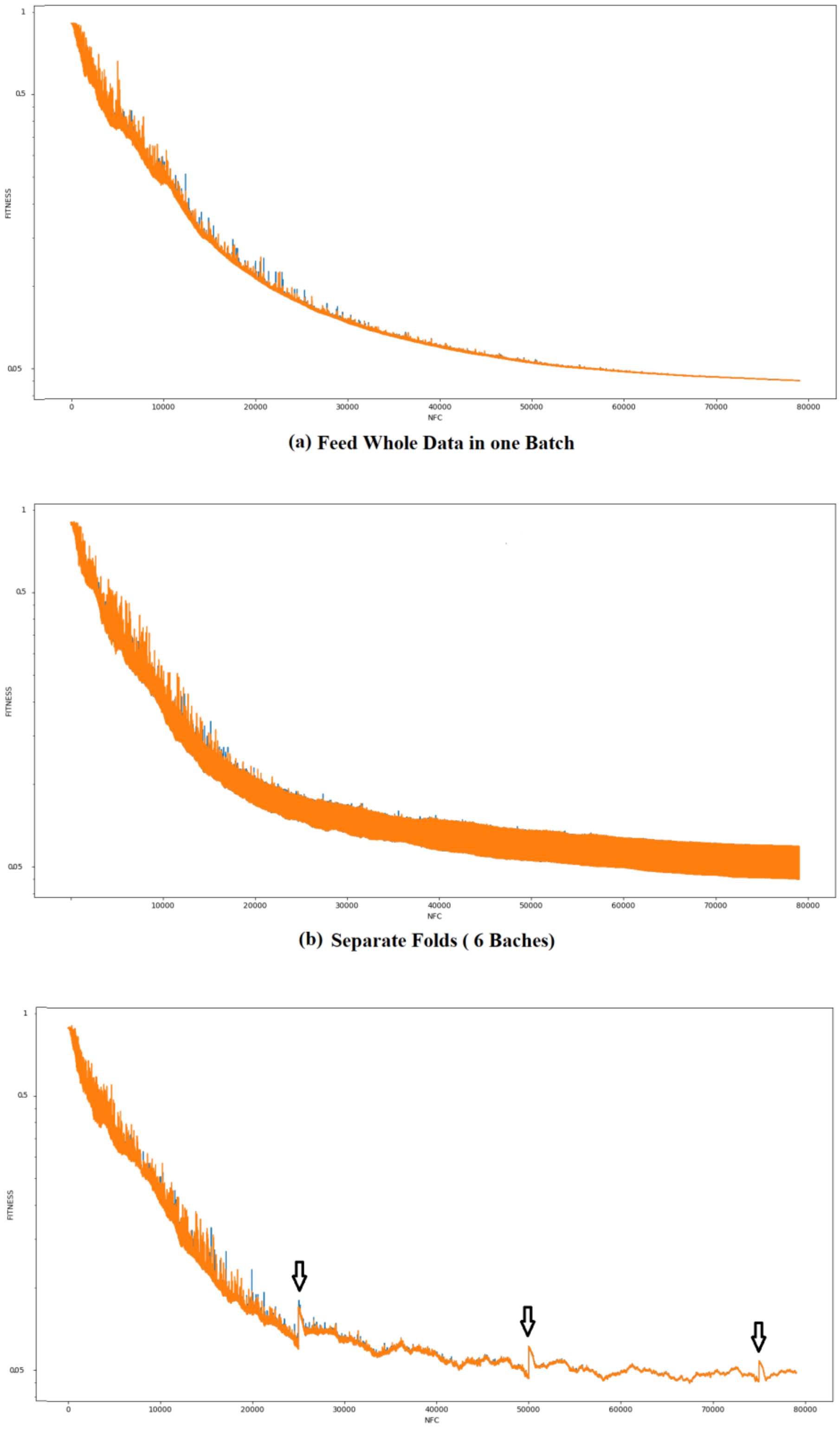


Figure 5.4: (a) Feed whole data, (b) Separate folds, (c) Sliding fold

Table 5.3: Different feeding data approaches

	Sliding Fold **(Sliding amount = 100)**	**Separate Folds**	**Feed Whole Data**
Fixed center initialization	93.52	93.71	94.66
Random center initialization **μ=0 σ=0.1**	93.7	93.82	**94.74**
Speed-up	~6 χ	~6 χ	1 χ



(a) Feed Whole Data in one Batch

(b) Separate Folds (6 Baches)

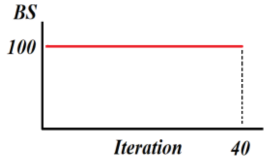
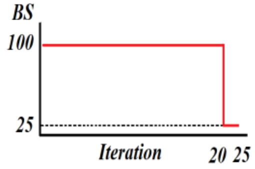
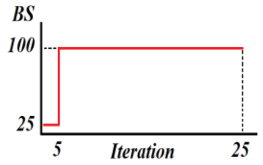
(c) Sliding Fold

Figure 5.5: Effect of different data feeding methods

5.6 Effect of Different Bundle Size (BS)

To find out the effect of the BS, we performed some tests by different bundle sizes. It is noticeable that the larger bundle size, leads to a lower NFC, and vice versa. For a fair comparison, the NFC for all tests are the same. So, the number of iterations varies based on the bundle size. If the BS is 100, the number of iterations is 40, but if the BS is 25, the number of iterations is decreased to ten. Table 5.3 represents the effect of a different BS. The BS can be fixed for all iterations or can be changed for different iterations. For example, for the first five iterations, the BS can be 25, and for 20 iterations, it can be 100. Having a low number of iterations leads to an immature convergence. Table 5.4 reports three test scenarios where the number of iterations is no less than 25.

Table 5.4: Effect of bundle size

Bundle Size (BS)			
Feed Separate Folds	93.82	93.95	95.23
Feed Whole data	94.74	94.88	**96.12**

Results show that when BS is low for the first iterations, accuracy is higher than other cases. It indicates better exploration, which is the effect of lower BS (BS=25 for five iterations), leading to higher accuracy. However, for accelerating the optimization process, we should increase the BS for other iterations (BS=100 for 20 iterations).

5.7 Comparing the results with SGD

In this section, the results of the proposed CS and SGD are compared. In sub-section 5.7.1, all training data is used, but in the second test (sub-section 5.7.2), a small subset (1/60 of the training data) of training data is used. This is to see the effect of training with a small size dataset, which is the case in many real-world applications, because there is no access to the large labeled training dataset.

5.7.1 Using all training data

For comparing the proposed scheme with SGD, which is well known for training neural networks, the network with 266,610 weights has been trained by both SGD and proposed CS. The training data is MNIST (handwriting digits). In this test, all training data (60,000) was used. The performance plot of training and testing can be seen in Figures 5.6 and 5.7, respectively. The proposed CS starts with better accuracy and converges to the optimum point faster than SGD, but in the end, SGD can also reach it. The results show almost the same test accuracy. So, the proposed method can be introduced as an alternative for training neural networks. THE proposed CS shows a remarkable ability for optimizing very large-scale problems. Results of training and testing are reported in table 5.5. In some cases, results on the MNIST is 100%, but these networks are more complicated CNN networks with many layers. The result reported here is the best result that can be achieved from this network.

In Figure 5.7, the dashed red line shows if the training process is stopped after five iterations which is equal to 10D fitness calls, the accuracy is relatively high (92.11%). This early stopping method is very beneficial in evolving neural network processes. Assume that a neural network hyperparameters or structure should be optimized as we did in one of our previous works [68]. Therefore, for each fitness call, the network should be trained. Many fitness calls are needed for the optimization process, so the network should be trained many times which is very time consuming. If early stopping leads to high accuracy, which is the case for the proposed CS method, the optimization process can be accelerated very much.

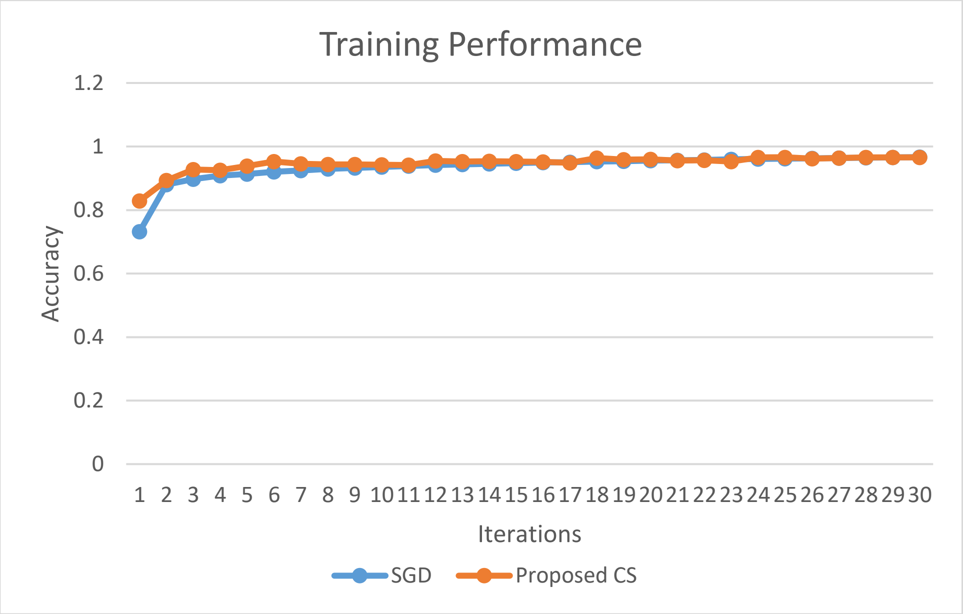


Figure 5.6: Training performance plot by feeding all training data

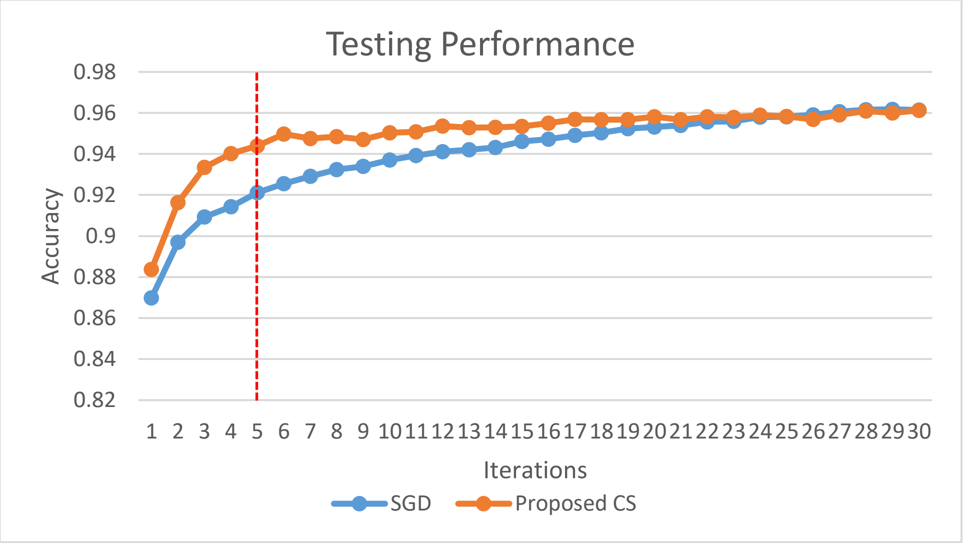


Figure 5.7: Testing performance plot by feeding all training data

Table 5.5: Comparing the results of the proposed CS and SGD by feeding all training data

NN Optimizer	SGD	Proposed CS
Train Accuracy	96.65	96.59
Test Accuracy	96.14	96.12

5.7.2 Using a small subset of training data

For performing this test, 1000 training samples from the MNIST Fashion dataset are used, which imitates the lack of labeled training data. This is one of the challenges of training networks and makes the problem harder for neural network to handle. The MNIST Fashion is a dataset with 60,000 training data samples, and 10,000 testing data samples in ten categories. Different categories and labels are shown in Table 5.6.

Table 5.6: Categories of data in MNIST Fashion

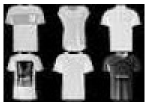
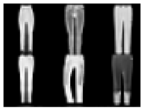
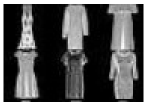
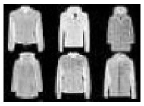
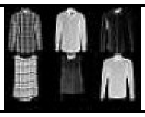
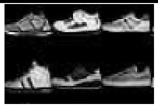
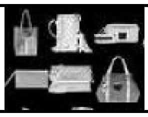
Labels	**0**	**1**	**2**	**3**	**4**
Description	T-shirt/Top	Trouser	Pullover	Dress	Coat
Samples					
Labels	**5**	**6**	**7**	**8**	**9**
Description	Sandals	Shirt	Sneaker	Bag	Ankle boot
Samples					

Table 5.7 reports test results for three test cases. The first one is SGD, the second one is Proposed CS, and the third one is a hybrid. CS has the best accuracy, and SGD cannot reach good accuracy with small size training data. In the hybrid method, only in the first iteration, the network is trained by CS. For other iterations (epochs), the weights which have been adjusted by CS are passed to the SGD as initial points. Then, SGD tries to adjust

the weights in the following epochs. Hybrid method benefits from both the advantages of CS, which is more accurate, and SGD, which is less time-consuming.

Table 5.7: Comparing SGD, Proposed CS and Hybrid optimizer by feeding small subset of training data

NN Optimizer	**SGD**	**Proposed CS**	**Hybrid**
Train Accuracy	75.2	**96.9**	86.2
Test Accuracy	71.1	**77.4**	75.7

Performance plots for training and testing are illustrated in Figure 5.8 and 5.9, respectively. The proposed CS can reach the solution with a small number of iterations. This fast convergence rate is very beneficial because optimizing weights in neural networks is demanding. So, a fast convergence rate reduces the time and computational cost of training.

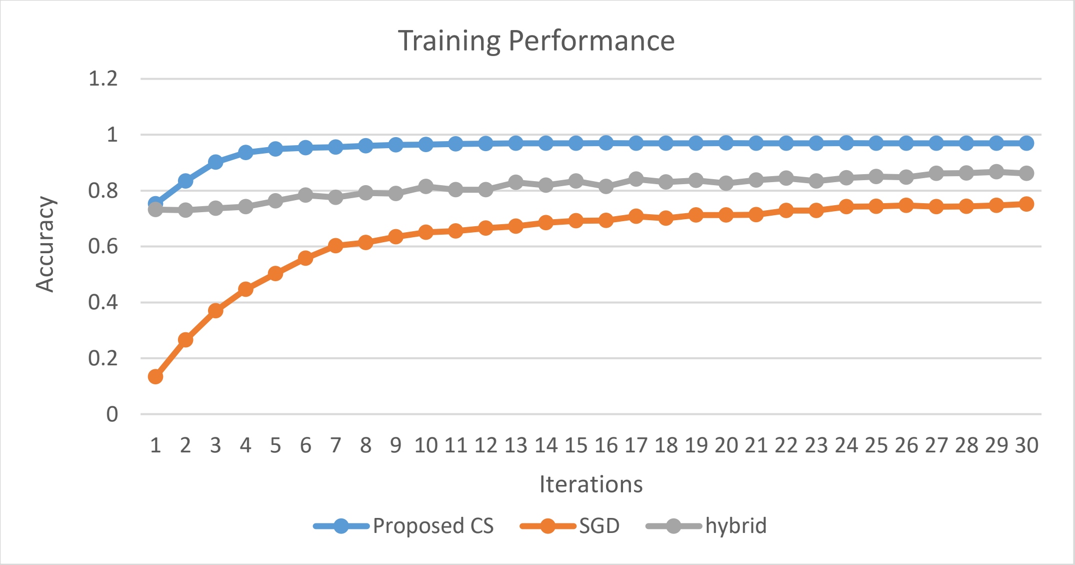


Figure 5.8: Training performance plot by feeding small subset of training data

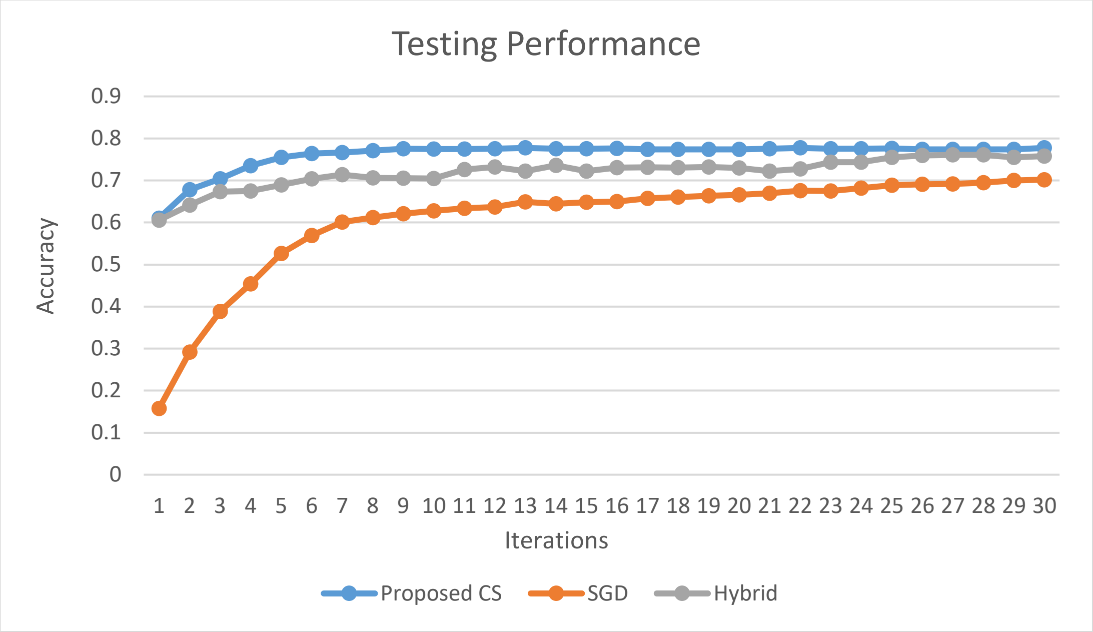
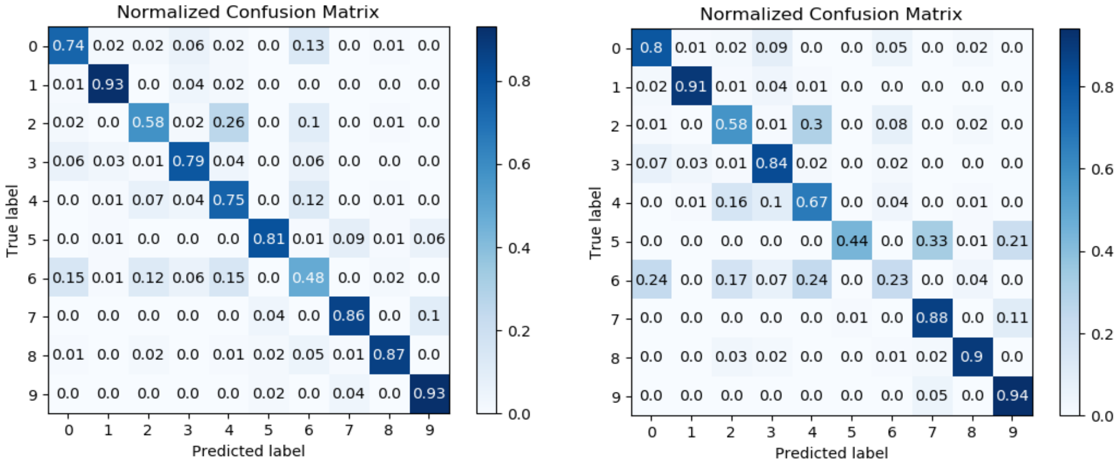


Figure 5.9: Testing the performance plot by feeding small subset of training data

Accuracy by itself cannot show the real performance of classification methods. Accuracy considers both precision and recall, but in some cases, recall is good. But precision is bad and vice versa. For having better vision towards network performance, confusion matrix, precision, and recall should be reported separately. The normalized confusion matrix for SGD and proposed CS is shown in Figure 6.10. For proposed CS, the number and amount of confusions between classes are less than SGD, which shows better classification ability. Table 5.8 reports precision and recall for each class. In both precision and recall cases, CS reaches better results. The dimension of the problem is not changed, and it is 266.610.

Table 5.8: Precision and recall for each class

Precision	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Average**
SGD	0.688	0.949	0.595	0.705	0.533	0.956	0.516	0.683	0.894	0.740	0.7259
Proposed CS	0.749	0.928	0.695	0.775	0.597	0.887	0.501	0.866	0.927	0.849	**0.7774**
Recall	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Average**
SGD	0.798	0.913	0.576	0.84	0.672	0.44	0.23	0.877	0.902	0.937	0.7185
Proposed CS	0.744	0.927	0.577	0.792	0.749	0.809	0.485	0.86	0.871	0.929	**0.7743**



(a) Proposed CS

(b) SGD

Figure 5.10: Confusion matrix for the proposed method and SGD

5.8 Conclusion

In this chapter, a new way of training the neural network based on two-extreme points CS was proposed. It shows that a very high-dimensional problem can be optimized by this method. Moreover, the lack of labeled training data, which is the case in many real-world applications, is in favor of the proposed method, and it performs better than other well-known methods. So, this algorithm can be a feasible alternative method to train neural networks. Moreover, proposed CS algorithms can find good solutions with very small NFC, which can be very beneficial in evolving neural network procedure.

This algorithm treats the problem as a black-box optimization task, so it has the ability to train complex network structures such as graph-based networks, which cannot be trained by GD techniques.

Chapter 6

Conclusion Remarks and Future Works

Based on the no-free-lunch theorem, there is no algorithm that works best for all problems. Table 6.1 demonstrates which algorithm performs best on specific domains. The proposed method has good performance in a wide variety of functions, which are high-dimensional.

Table 6.1: Effective domains of optimization methods

GD	**Metaheuristic/swarm**	**Proposed CS**
Differentiable	Differentiable/ non-differentiable	Differentiable/ non-differentiable
Convex (or can be considered as convex)	Convex/non-convex	Convex/non-convex
Smooth	Smooth/non-smoosh	Smooth/non-smoosh
Low to very high-dimensional	Not very high-dimensional	High to very high-dimensional

In this work, a specific domain of optimization was chosen, which is a non-convex-large-scale black-box area. In this domain, it is hard to find a feasible method for optimization problems. We developed an optimization method based on a coordinate search algorithm step by step. The best one, which is the two-extreme coordinate search, shows remarkable performance in front of four state-of-the-art algorithms in five functions of the CEC-2013 benchmarks. This algorithm also benefits from the expansion option, which expands the search region in some cases to help the optimizer perform better exploration.

As a case study, we used two-extreme CS for training a fully connected neural network with more than a quarter of a million weights on a personal computer with a single CPU. In order to address the convergence speed issue, a bundling technique based on LHS has been deployed to group the variables and treats each group as one variable to speed up the optimization process. It shows good results in front of the SGD algorithm, which is one of the most powerful algorithms for training neural networks. For situations with lack of labeled training data, the proposed CS method performs better than SGD. The performance plots show a high convergence rate, which reveals the ability of the proposed method to find a reasonable solution with lower NFC. This ability is remarkable in the expensive optimization process where each of their FC comes with high costs in terms of time, computation, material, and so on. This ability can be very beneficial in evolving neural network procedure, and helps accelerate the procedure, which is very crucial for evolving processes.

The proposed CS has been used in all cases as a black-box optimizer. This means we do not tailor the optimizers for a specific problem. Using an optimizer for black-box optimization shows it can be used for many applications without using metamodels or any kind of approximate models. Also, it can be used very easily, and there is no need to calculate complex mathematical functions as a fitness function. Moreover, in some cases such as graph-based neural networks, back propagation cannot be applied, so the only way to train them is limited to non-gradient based methods such as the proposed CS.

Simplicity is one of the major key factors in designing an optimizer. A simple optimizer can be leveraged in many applications. Moreover, it is easy to understand, so users who are not experts in optimization techniques can benefit from it in their applications and process. On the contrary, very complex algorithms are less likely to be used by others. Proposed CS algorithm is simple and can be applied in a few lines of code.

Future works

The proposed algorithm is in early stages of evolution. It shows remarkable capabilities, but there is a lot of room for improvement. In this sub-section, potential improvements include:

a- Each algorithm has some flow and some strength. By hybridization of different algorithms, its strength can be elevated. For example, we used CS and SGD for training networks (in Chapter 5) in a hybrid manner, which shows good performance and high-speed operation. Different hybridization schemes can be used to get better performance or speed-up. Moreover, in multilevel optimization, CS can be combined with other methods to achieve better results.

b- Partial function evaluation in CS algorithms can be used to speed up optimization. CS, unlike metaheuristic, swarm, and GD, works with partial evaluation. In cases where we struggle with partial evaluation, such as training the last layers of pretrained networks, SC can speed-up the process to the factor of D. Instead of calculating all components of the fitness function, we calculate one component while other components are fixed.

c- CS can be leveraged for Training graph-based networks. Graph-based networks cannot be trained by gradient-based algorithms because there is no straightforward way for backpropagation. Proposed CS can optimize black-box problems, so it can be used for such networks without the need to calculate the fitness function, and there is no need to struggle with other complexities of graph-based networks.

References

[1] N. Andréasson, A. Evgrafov, and M. Patriksson, "An introduction to optimization: Foundations and fundamental algorithms," *Chalmers University of Technology Press: Gothenburg, Sweden,* vol. 1, pp. 1-205, 2005.

[2] A. J. Zaslavski, "Convex Optimization with Computational Errors," *Springer*, 2020.

[3] P. B. Dimitri, N. Angelia, and E. Asuman, "Convex analysis and optimization," *Athena Scientific: Belmont, Massachusetts United States,* 2003.

[4] P. M. Pardalos, A. Žilinskas, and J. Žilinskas, "Non-convex multi-objective optimization," *Springer*, 2017.

[5] A. Jameson, "Gradient based optimization methods," *MAE Technical Report No,* no. 2057, 1995.

[6] A. N. Tikhonov, A. Goncharsky, V. Stepanov, and A. G. Yagola, "Numerical methods for the solution of ill-posed problems," *Springer Science & Business Media*, 2013.

[7] J. R. Sampson, "Adaptation in natural and artificial systems," *ed: Society for Industrial and Applied Mathematics*, 1976.

[8] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization,* vol. 11, no. 4, pp. 341-359, 1997.

[9] X. Wang, R. D. Haynes, and Q. Feng, "A multilevel coordinate search algorithm for well placement, control and joint optimization," *Computers & Chemical Engineering,* vol. 95, pp. 75-96, 2016.

[10] E. Frandi and A. Papini, "Coordinate search algorithms in multilevel optimization," *Optimization Methods and Software,* vol. 29, no. 5, pp. 1020-1041, 2014.

[11] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods," *SIAM review,* vol. 45, no. 3, pp. 385-482, 2003.

[12] I. Loshchilov, M. Schoenauer, and M. Sebag, "Adaptive coordinate descent," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 885-892.

[13] E. Tzinis, "Bootstrapped Coordinate Search for Multidimensional Scaling," *arXiv preprint arXiv:1902.01482,* 2019.

[14] W. Huyer and A. Neumaier, "Global optimization by multilevel coordinate search," *Journal of Global Optimization,* vol. 14, no. 4, pp. 331-355, 1999.

[15] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, 1995, vol. 4: Citeseer, pp. 1942-1948.

[16] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, 1999, vol. 2: IEEE, pp. 1470-1477.

[17] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation,* vol. 1, no. 1, pp. 67-82, 1997.

[18] D. Newton, F. Yousefian, and R. Pasupathy, "Stochastic gradient descent: recent trends," in *Recent Advances in Optimization and Modeling of Contemporary Problems*: INFORMS, 2018, pp. 193-220.

[19] M. Pandey and B. Pal, "Adaptation to best fit learning rate in batch gradient descent," *International Journal of Science and Research (IJSR),* vol. 3, no. 8, 2014.

[20] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*: Springer, 2012, pp. 421-436.

[21] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983,* 2016.

[22] O. Shamir and T. Zhang, "Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes," in *International conference on machine learning*, 2013, pp. 71-79.

[23] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming,* vol. 151, no. 1, pp. 3-34, 2015.

[24] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications,* vol. 109, no. 3, pp. 475-494, 2001.

[25] Z.-Q. Luo and P. Tseng, "On the convergence of the coordinate descent method for convex differentiable minimization," *Journal of Optimization Theory and Applications,* vol. 72, no. 1, pp. 7-35, 1992.

[26] Z.-Q. Luo and P. Tseng, "Error bounds and convergence analysis of feasible descent methods: a general approach," *Annals of Operations Research,* vol. 46, no. 1, pp. 157-178, 1993.

[27] D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and distributed computation: numerical methods," *Prentice hall Englewood Cliffs,* NJ, 1989.

[28] P. Tseng and S. Yun, "A coordinate gradient descent method for linearly constrained smooth optimization and support vector machines training," *Computational Optimization and Applications,* vol. 47, no. 2, pp. 179-206, 2010.

[29] S. Yun and K.-C. Toh, "A coordinate gradient descent method for ℓ 1-regularized convex minimization," *Computational Optimization and Applications,* vol. 48, no. 2, pp. 273-307, 2011.

[30] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming,* vol. 117, no. 1-2, pp. 387-423, 2009.

[31] T. T. Wu and K. Lange, "Coordinate descent algorithms for lasso penalized regression," *The Annals of Applied Statistics,* vol. 2, no. 1, pp. 224-244, 2008.

[32] M. Gürbüzbalaban, A. Ozdaglar, N. D. Vanli, and S. J. Wright, "Randomness and permutations in coordinate descent methods," *Mathematical Programming,* pp. 1-28, 2019.

[33] M. Gurbuzbalaban, A. Ozdaglar, P. A. Parrilo, and N. Vanli, "When cyclic coordinate descent outperforms randomized coordinate descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 6999-7007.

[34] Z. Z. Bai and W. T. Wu, "On greedy randomized coordinate descent methods for solving large linear least-squares problems," *Numerical Linear Algebra with Applications,* vol. 26, no. 4, p. e2237, 2019.

[35] I. Necoara, "Random coordinate descent algorithms for multi-agent convex optimization over networks," *IEEE Transactions on Automatic Control,* vol. 58, no. 8, pp. 2001-2012, 2013.

[36] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization,* vol. 22, no. 2, pp. 341-362, 2012.

[37] J. Chen and Q. Gu, "Accelerated Stochastic Block Coordinate Gradient Descent for Sparsity Constrained Nonconvex Optimization," in *UAI*, 2016.

[38] T. Glasmachers and U. Dogan, "Accelerated coordinate descent with adaptive coordinate frequencies," in *Asian Conference on Machine Learning*, 2013, pp. 72-86.

[39] E. Song, Z. Shen, and Q. Shi, "Block coordinate descent almost surely converges to a stationary point satisfying the second-order necessary condition," *optimization-online,* 2017.

[40] J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao, "Block coordinate descent for deep learning: Unified convergence guarantees," *arXiv preprint arXiv:1803.00225,* 2018.

[41] R. Liu, W.-C. Sun, T. Hou, C.-H. Hu, and L.-B. Qiao, "Block coordinate descentwith time perturbation for nonconvex nonsmooth problems in real-world studies," *Frontiers of Information Technology & Electronic Engineering,* vol. 20, no. 10, pp. 1390-1403, 2019.

[42] P. Latafat, A. Themelis, and P. Patrinos, "Block-coordinate and incremental aggregated proximal gradient methods for nonsmooth nonconvex problems," *arXiv,* p. arXiv: 1906.10053, 2019.

[43] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, "Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation," *John Wiley & Sons*, 2009.

[44] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology),* vol. 68, no. 1, pp. 49-67, 2006.

[45] H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin, "A primer on coordinate descent algorithms," *arXiv preprint arXiv:1610.00040,* 2016.

[46] T. T.-K. Lau, J. Zeng, B. Wu, and Y. Yao, "A proximal block coordinate descent algorithm for deep neural network training," *arXiv preprint arXiv:1803.09082,* 2018.

[47] H. Liu, Y. Wang, and N. Fan, "A hybrid deep grouping algorithm for large scale global optimization," *IEEE Transactions on Evolutionary Computation,* 2020.

[48] R. G. Regis and C. A. Shoemaker, "Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization," *Engineering Optimization,* vol. 45, no. 5, pp. 529-555, 2013.

[49] L. Kallel and M. Schoenauer, "Alternative Random Initialization in Genetic Algorithms," in *ICGA*, 1997: Citeseer, pp. 268-275.

[50] G. Thimm and E. Fiesler, "Neural network initialization," in *International Workshop on Artificial Neural Networks*, 1995: Springer, pp. 535-542.

[51] S. Rahnamayan and G. G. Wang, "Toward effective initialization for large-scale search spaces," *Trans Syst,* vol. 8, no. 3, pp. 355-367, 2009.

[52] M. S. Maučec and J. Brest, "A review of the recent use of Differential Evolution for Large-Scale Global Optimization: An analysis of selected algorithms on the CEC 2013 LSGO benchmark suite," *Swarm and Evolutionary Computation,* vol. 50, p. 100428, 2019.

[53] G. Wu, R. Mallipeddi, and P. N. Suganthan, "Ensemble strategies for population-based optimization algorithms–A survey," *Swarm and evolutionary computation,* vol. 44, pp. 695-711, 2019.

[54] V. Nissen and J. Propach, "On the robustness of population-based versus point-based optimization in the presence of noise," *IEEE Transactions on Evolutionary Computation,* vol. 2, no. 3, pp. 107-119, 1998.

[55] A. A. B. Davood Zaman Farsa, Ehsan Rokhsatyazdi, Shahryar Rahnamayan, "Population-Based Coordinate Descent Based on Majority Voting," *Under Revision*, 2020.

[56] A. A. Hadi, A. W. Mohamed, and K. M. Jambi, "LSHADE-SPA memetic framework for solving large-scale optimization problems," *Complex & Intelligent Systems,* vol. 5, no. 1, pp. 25-40, 2019.

[57] A. LaTorre, S. Muelas, and J.-M. Peña, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," in *2013 IEEE congress on evolutionary computation*, 2013: IEEE, pp. 2742-2749.

[58] F. Wei, Y. Wang, and T. Zong, "Variable grouping based differential evolution using an auxiliary function for large scale global optimization," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014: IEEE, pp. 1293-1298.

[59] D. Molina and F. Herrera, "Iterative hybridization of DE with local search for the CEC'2015 special session on large scale global optimization," in *2015 IEEE congress on evolutionary computation (CEC)*, 2015: IEEE, pp. 1974-1978.

[60] J. Zhou, Z. Duan, Y. Li, J. Deng, and D. Yu, "PSO-based neural network optimization and its utilization in a boring machine," *Journal of Materials Processing Technology,* vol. 178, no. 1-3, pp. 19-23, 2006.

[61] M. Carvalho and T. B. Ludermir, "Particle swarm optimization of neural network architectures andweights," in *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, 2007: IEEE, pp. 336-339.

[62] E. A. Grimaldi, F. Grimaccia, M. Mussetta, and R. Zich, "PSO as an effective learning algorithm for neural network applications," in *Proceedings. ICCEA 2004. 2004 3rd International Conference on Computational Electromagnetics and Its Applications, 2004.*, 2004: IEEE, pp. 557-560.

[63] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Processing Letters,* vol. 17, no. 1, pp. 93-105, 2003.

[64] Y. Lu and R. Yang, "Not All Features Are Equal: Feature Leveling Deep Neural Networks for Better Interpretation," *arXiv preprint arXiv:1905.10009,* 2019.

[65] J. Cheng and M. J. Druzdzel, "Latin hypercube sampling in Bayesian networks," in *FLAIRS Conference*, 2000, pp. 287-292.

[66] M. D. Shields and J. Zhang, "The generalization of Latin hypercube sampling," *Reliability Engineering & System Safety,* vol. 148, pp. 96-108, 2016.

[67] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612,* 2018.

[68] S. R. Ehsan Rokhsatyazdi, Hossein Amirinia, Sakib Ahmed, "Optimizing LSTM Based Network For Forecasting Stock Market," *presented at the WCCI IEEE*, 2020.