REAL PARAMETER OPTIMIZATION USING DIFFERENTIAL EVOLUTION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Deepak Dawar

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

REAL PARAMETER OPTIMIZATION USING DIFFERENTIAL EVOLUTION

By

Deepak Dawar

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Simone Ludwig
Chair

Kendall Nygard

Cheryl Wachenheim

Approved:

10/29/2013	Brian Slator
Date	Department Chair

ABSTRACT

Over recent years, Evolutionary Algorithms (EA) have emerged as a practical approach to solve hard optimization problems presented in real life. The inherent advantage of EA over other types of numerical optimization methods lies in the fact that they require very little or no prior knowledge of the objective function. Information like differentiability or continuity is not necessary. The inspiration to learn from evolutionary processes and emulate them on a computer comes from varied directions, the most pertinent of which is the field of optimization. This paper presents one such Evolutionary Algorithm known as Differential Evolution (DE) and tests its performance on benchmark problems. Different variants of basic DE are discussed and their advantages and disadvantages are listed. This paper, through exhaustive experimentation, proposes an acceptable set of control parameters which may be applied to most of the benchmark functions to achieve good performance.

ACKNOWLEDGEMENTS

I would like to convey my earnestness to my primary advisor and mentor, Dr. Simone Ludwig for her enthusiastic and encouraging support during the pursuit of my research. This defense paper is as much as a product of her guidance as the work of my own research. I also express my sincere regards to Dr. Nygard for being an advisor and coach at the same time and for giving his timely inputs to steer my research in the right direction. Lastly, I convey my heartfelt appreciation for Dr. Cheryl Wachenheim for being on my supervisory committee and providing her statistical inputs on the results and plots.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

1. INTRODUCTION

An optimization process/algorithm, given a set of constraints, tries to find the best solution from all feasible solutions for a given problem. In real life though, there remain a large number of problems in class NP (non-deterministic polynomial) [1] for which finding the best solution is not possible in polynomial time, at least as of now. In such cases, it is more plausible to find a good enough solution (sub-optimal) instead of spending a great deal of computational time to find the best solution. Moreover, optimization is not always about finding the perfect solution. In those scenarios, it is more about finding a good solution given a set of constraints or environment. As the constraints change the solution also needs to change accordingly. This has many parallels in the Evolutionary Process.

Evolution in a way is also an optimization process wherein the solution or adaptive behavior depends upon the constraints posed by the environment of an organism. The behavior of an organism is optimized and changes according to the environment. Since evolution has been able to produce organisms of high perfection over a long period of time, there is always a big motivation behind application of evolutionary principles in the optimization process to solve hard real world engineering problems. It was this motivation that Evolutionary Algorithms (EA) came on the horizon of optimization.

EA are stochastic optimization algorithms which imitate biological processes that allow a population of organisms to adapt to its environment using the principles of mutation, crossover and survival of the fittest. As can be inferred from the evolutionary process that it is more about adaptation than perfection, it lends itself naturally to the optimization process. Thus it would be wise to describe this evolutionary process in an algorithmic form to find solutions to difficult optimization problems.

Differential Evolution (DE) is a simple yet powerful evolutionary algorithm (EA) for global optimization introduced by Price and Storn [2]. The DE algorithm has gradually become more popular among other EA and has been used in many practical cases, mainly because it has demonstrated good convergence properties and is easy to understand [3].

Some of the reasons why DE has emerged as an attractive optimization tool are as follows:

1. DE is simple and easy to implement as compared to other EA. Simplicity is a valuable attribute and added advantage of any algorithm for application by practitioners in other fields.
2. Despite its simplicity DE shows robust performance in comparison with several other EA on a wide variety of problems including unimodal, multimodal, separable, non-separable ones [4].
3. The number of control parameters in DE is few.

In this paper the classic DE algorithm, commonly known as (*DE/rand/1/bin*), is evaluated on 20 IEEE CEC benchmark functions at three different problem dimensionalities (*10, 30, 50*). While optimizing a real world problem with DE, it is very time consuming to test the objective function exhaustively with all the good control parameter settings. Thus, there is a need for statistical and empirical studies which may suggest a set of rules to choose good parameter settings for a given function landscape or properties.

It is with this intent, in this paper, an exhaustive set of control parameters are experimented upon, on a multitude of benchmark functions to deduce good parameter settings for a large set of functions.

This paper is composed of six chapters which are as follows:

Chapter 1 introduced the basic DE algorithm and reasons for its popularity.

Chapter 2 describes DE in detail, lists its variants and discusses the importance of control parameters on the performance of the algorithm.

Chapter 3 presents the implementation details of the algorithm which include the pseudo-code and a brief overview of the related JAVA classes.

Chapter 4 presents the IEEE CEC 2013 Test Suite and lists the first 20 benchmark functions. The experimental setup to test the DE performance on the test suite follows later in the chapter.

Chapter 5 presents and discusses the results obtained. The results for each dimensionality are presented with data statistics that include the Best, Worst, Mean and Standard-Deviation values obtained for every benchmark function evaluated.

Chapter 6 concludes the paper and a set of control parameters of DE is proposed that may be applied to a large number of problems as a rule of thumb.

2. DIFFERENTIAL EVOLUTION: BASIC CONCEPTS

DE is a simple real parameter optimization algorithm. It is a direct search method that utilizes a pre-specified number of random solutions and continuously improves them through a series of mutations and re-combinations, as the search progresses. The number of pre-specified solutions does not change during the lifetime of the classical DE algorithm. It works through a cycle of stages as represented in Figure 1. The algorithm is explained in the next section.

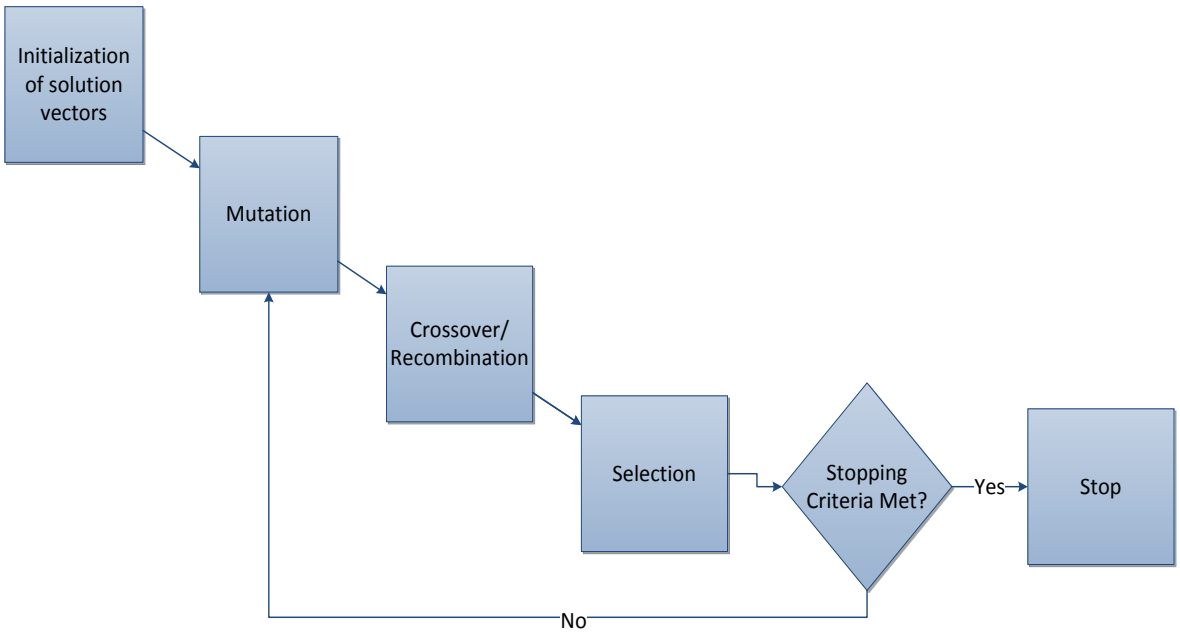

Figure 1. Steps in Differential Evolution

2.1. Initialization

DE searches for a global optimum in a D dimensional real parameter space R^D. It starts with a randomly generated population of *NP D* dimensional real-valued parameter individuals, where *NP* represents the number of individuals. In broadly accepted DE terminology, each individual candidate solution is called a vector. The vectors are changed over generations, denoted by *G =0, 1, 2…,*G_{max}. Thus the i^{th} vector solution in the population can be represented as

$$X_j^i = [x_1^i, x_2^i, x_3^i, ..., x_D^i] \quad (1)$$

Every parameter x_j^i in a given vector has a specific range within which it has to be restricted, denoted by x_{jmin}^i and x_{jmax}^i. The initial population should cover this range by randomly initializing the solution vectors while making sure that none of the parameters are initiated outside the bound. Hence we may initialize the j^{th} component of the i^{th} vector as

$$x_j^i = x_{jmin}^i + random_j^i\,[0, 1] \times (x_{jmax}^i - x_{jmin}^i) \quad (2)$$

where $random_j^i\,[0, 1]$ is a uniformly distributed random number lying between 0 and 1 and is instantiated independently for each j^{th} component of the i^{th} vector [4].

2.2. Mutation

Biologically, mutation is a change in the gene characteristics of a chromosome. Applied to evolutionary computation it means a change in the parameters of the vector through a perturbation with a random element. In DE literature, a parent vector from the current generation is called *target* vector, the mutant vector obtained through differential perturbation is called *donor* vector and the offspring obtained through recombination of *target* and *donor* is called *trial* vector. In classic DE, to create a donor vector for each i^{th} *target* vector from the current population, three other distinct vectors, say X_{r1}^i, X_{r2}^i and X_{r3}^i are selected randomly from the current population. The indices r_1, r_2 and r_3 are mutually exclusive integers randomly chosen from the range [1, *NP*], which are also different from the base vector index *i*. These indices are randomly generated once for each mutant vector. Now the difference of any two of these three vectors is multiplied by a scalar number called scaling factor *F* and the scaled difference is added to the third vector to obtain the donor vector V_G^i [4]. We can express the relation as

$$V_G^i = X_G^{r1} + F \times (X_G^{r2} - X_G^{r3}) \quad (3)$$

2.3. Crossover

Diversity is an important aspect of evolution. After the generation of *donor* vector through mutation, a crossover takes place wherein parameters from *target* vector and *donor* vector are selected based on some probability distribution to form a *trial* vector $U_G^i = [U_{1,G}^i , U_{2,G}^i, U_{3,G}^i, ..., U_{D,G}^i]$. The two most popular types of crossover methods are *exponential and binomial* [5]. In exponential crossover we randomly choose an integer *n* between [1, D]. This integer *n* determines the index from where the crossover operation would start. Another integer *K* is chosen from the same interval which denotes the number of parameters that would contribute to *trial* vector. The trial vector is obtained as

$$u_{j,G}^i = \begin{cases} v_{j,G}^i & \text{for } j = n, n+1 \ldots n+K-1 \\ x_{j,G}^i & \text{for all other } j \in [1, \mathrm{D}] \end{cases} \quad (4)$$

The integer *K* is drawn from [1, D] according to the following pseudo-code [4].

K = 0;

do

{

K = *K* + 1;

} while ((random (0, 1) ≤ *Cr*) AND (*K* ≤ *D*)).

"Cr" is called the crossover rate.

Binomial Crossover differs from the exponential counterpart in the sense that it is performed on each of the D parameters whenever a randomly generated number between the interval [0, 1] is less than or equal to the *Cr* value. Thus, the parameters inherited by the *trial* vector from the *donor* vector have a binomial distribution. This could be written as:

$$u_{j,G}^i = \begin{cases} v_{j,G}^i & \text{if } (random_j^i [0,1] \leq Cr \text{ or } j = j_{rand}) \\ x_{j,G}^i & Otherwise \end{cases}$$

Where $random_j^i$ [0, 1] is a randomly generated real number between the interval [0, 1] and is generated anew for every j^{th} parameter of the solution vector and j_{rand} is a randomly chosen index between [1, D] to ensure that *trial* vector gets at least one element from the *donor* vector.

2.4. Selection

This is the operation that determines whether the *offspring* which is represented by the *trial* vector or the *parent* which is represented by *target* vector would survive to the next generation. The procedure is simple. If the fitness of the *trial* vector is better or at least equal to the *target* vector, it moves to the next generation i.e., *G= G+1* else *target* vector is promoted. Choosing the *trial* vector over the *target* vector even if they have the same fitness, allows the vectors to explore the flat spaces better.

$$X_{G+1}^i = U_G^i \quad \text{if } f(U_G^i) \leq f(X_G^i)$$

$$= X_G^i \quad \text{if } f(U_G^i) > f(X_G^i) \quad \text{where } f(X) \text{ is the objective function.}$$

It should be noted that in any case, the population size remains the same, i.e., the generation *G+1* would have the same number of vectors as was in *G*.

2.5. Classic DE and Its Variants

DE algorithm in its fundamental form uses a randomly chosen vector X_1 and one weighted difference $F(X_2\text{-}X_3)$ to perturb the chosen vector. This mutation strategy is known as *DE/rand/1*, where *DE* stands for Differential Evolution, *rand* means that the vectors $[X_1, X_2, X_3]$ are randomly chosen, and 1 suggests that only one weighted difference is used. When this mutation strategy is used with binomial crossover, it is referred to as *DE/rand/1/bin*. This representation gives us an idea as to how the different schemes are named. The other four prominent schemes suggested by Storn and Price [2, 5] are:

"DE/best/1":

$$V_G^i = X_{best,G}^i + F \times (X_{r2,G}^i - X_{r3,G}^i)$$

"DE/target-to-best/1":

$$V_G^i = X_{target,G}^i + F \times (X_{best,G}^i - X_{target,G}^i) + F \times (X_{r1,G}^i - X_{r2,G}^i)$$

"DE/best/2":

$$V_G^i = X_{best,G}^i + F \times (X_{r1,G}^i - X_{r2,G}^i) + F \times (X_{r3,G}^i - X_{r4,G}^i)$$

"DE/rand/2":

$$V_G^i = X_{r1,G}^i + F \times (X_{r2,G}^i - X_{r3,G}^i) + F \times (X_{r4,G}^i - X_{r5,G}^i)$$

The indices r_1, r_2, r_3, r_4 and r_5 are mutually exclusive, randomly chosen and are different from the *base/target* index *i*. $X_{best,G}^i$ is the vector with best fitness in the generation *G*. *F* is the scaling factor already described above.

Each mutation strategy described above may be combined with any of the 2 crossover strategies resulting in 5 × 2 = 10 variants. None of the variants listed above is regarded as the best to solve all the problems. Every variant has it is own advantages and disadvantages. For example *DE/best/1* usually converges very fast as it progresses along the best vector in the population but leads to quick saturation of the population. In contrast *DE/rand/1* is relatively slow but maintains good diversity throughout the optimization process.

2.6. A Discussion on Control Parameters of DE

As already mentioned, the popularity of DE is attributed to its operational simplicity owing to its few control parameters namely *NP* (population size), *F* (scaling factor), and *Cr*

(crossover rate). These control parameters have an exceedingly vital impact on searching capability and speed of convergence of the DE on a particular problem. For example, DE is quite sensitive to the initialization of the original population, not just its size [6]. Thus, the task of setting the best control parameters for a given problem is mostly trial and error based, tedious and may require an experienced hand with a notable background in art of fine tuning of parameters. There have been many studies aimed at finding a universal set of control parameters but the recommendations that came out of these studies were vague and sometimes contradictory [2, 8]. This paper attempts to perform a sensitivity analysis of the DE algorithm on CEC 2013 benchmark functions to find out the most applicable set of control parameters through exhaustive experimentation.

3. IMPLEMENTATION

3.1. Pseudo-Code

The classic DE algorithm consists of four basic steps as described in Chapter 2. One of the integral parts of any algorithm is specification of its stopping criteria. For DE, it can be defined in various ways, some of which are:

1. Running the algorithm for a fixed number of generations, Gmax.
2. Stopping the algorithm when no appreciable change is evident over a chosen number of generations.
3. Stopping when a pre-specified value of the objective function is reached.

Presented in Figure 2 is the pseudo-code for the DE algorithm.

---

1. *Read F, Cr and NP, G(no. of generations)*
2. *Randomly initialize the population of NP vectors as P= {* $X_{1,G}$, $X_{2,G}$ ... $X_{NP,G}$*} with each vector* $X_{i,G}$ = [X^i_{1G}, $X^i_{2,G}$ ... $X^i_{D,G}$]*, distributed uniformly between its min and max value represented as* X_{min} = [$x_{1,min}$, $x_{2,min}$ ... $x_{D,min}$] *and* X_{max}= [$x_{1,max}$, $x_{2,max}$ ... $x_{D,max}$] *where i ∈[1,NP].*
3. *While the stopping condition for the algorithm is not met*

   Do

   For each vector from 1 to NP

   3.1 Perform Mutation i.e. generate a donor V^i_G *for each target vector* X^i_G*, as follows*

   $$V^i_G = X^i_{r1,G} + F \times (X^i_{r2,G} - X^i_{r3,G})$$

   3.2 Perform Crossover i.e. generate a trial vector U^i_G *for each target vector* X^i_G *as follows*

   $$u^i_{j,G} = \begin{cases} v^i_{j,G} & \text{if } (random^i_j\ [0, 1] \leq Cr \text{ or } j = j_{rand}) \\ x^i_{j,G} & \text{Otherwise} \end{cases}$$

3.3 Perform Selection i.e. compare the fitness of trial and target vector and choose the more deserving candidate as follows

$X_{G+1}^{i} = U_{G}^{i}$ *if* $f(U_{G}^{i}) \leq f(X_{G}^{i})$

$= X_{G}^{i}$ *if* $f(U_{G}^{i}) > f(X_{G}^{i})$

$where\ f(X) is\ the\ objective\ function.$

$end\ for$

3.4 Increase the generation count, set generation G+1 as G.

end while

Figure 2. Pseudo-code for classical DE algorithm (DE/rand/1/bin)

The algorithm is implemented in JAVA and executed on a LINUX Debian 3.2.0-4 machine having Intel(R) Xeon(R) CPU X5680 @ 3.33GHz.

3.2. Source Code Overview

The DE algorithm has been implemented in JAVA programing language. The package consists of six core classes and a property file. The kernel is implemented as DEvolution.java. A user can specify the benchmark function, the mutation strategy and the crossover strategy in the property file named DE.properties. Any class that represents the benchmark function can be plugged into the system. The kernel reads the arguments from the properties file, runs the algorithm and reports the results. Snippets of the core classes are listed in Figures 3-6.

```
//constructor 1
//Initiate the benchmark function, mutation and crossoverStrategy
public DEvolution(String function,String mutationStrategy,String crossoverStrategy){

    try{
        this.function = function;
        Class<?>    clsFunc         =   Class.forName(this.function);/*Load the class that represents the function
                                                                        to be tested*/
        this.mutationStrategy       =   mutationStrategy;
        this.initialStrategy        =   this.mutationStrategy;
        this.crossoverStrategy      =   crossoverStrategy;
        /* create an instance of the class that represents the
        function and get the min and max values*/
        this.objectiveFunc              =   (Function)clsFunc.newInstance();
        Field       min                 =   clsFunc.getField("MIN");
        Field       max                 =   clsFunc.getField("MAX");
        Field       GLBMIN              =   clsFunc.getField("GLBMIN");
        this.MIN                        =   min.getDouble(this.objectiveFunc);
        this.MAX                        =   max.getDouble(this.objectiveFunc);
        this.GLBMIN                     =   GLBMIN.getDouble(this.objectiveFunc);
        System.out.println(" Min is "+ MIN + " and strategy is "+ this.mutationStrategy);
        this.initializePopulation(this.MIN, this.MAX);

        /*load the class that represents the
        *   mutation strategy
        *   specified in DE.properties.
        */
        try{
            Class<?>        clsMutate       =    Class.forName(mutationStrategy);
            this.ms                         =   (MutationStrategy)clsMutate.newInstance();
        }catch(ClassNotFoundException e){
            System.out.println("No mutation srtategy listed. Exiting!!!!");
            e.printStackTrace();
            System.exit(0);
        }
```

Figure 3. DEvolution.java

DEvolution is the main class of the algorithm. It maintains the global population which is manipulated by mutation and crossover strategies. It also keeps track of data statistics like selection percentage of the vectors through generations etc.

Major Responsibilities:

1. Initialization of the population.
2. Calling mutation, crossover and selection strategies.
3. Reporting the results.

```
public class Rand1 implements MutationStrategy{

    int i,j,k, pointer;
    double  temp[]          =   new double[DEvolution.DIMENSION];
    Random  rm              =   new Random();

    public void mutate(int target, int indexOfBestVector, double MIN, double MAX){


        //System.out.println("donor length in mutate before is "+ donor.length);
        while( (i = rm.nextInt(DEvolution.NP)) == target ){
            continue;
        }/*generate i and make sure it is not eaual to target*/

        while((j=rm.nextInt(DEvolution.NP)) == i && j == target){ /*generate j and make sure it is not equal to i or target*/
            continue;
        }

        while(true){/*generate k and make sure it is not equal to either i or j or target*/
            k=rm.nextInt(DEvolution.NP);
            if(k!=i && k!=j && k!=target)
                break;
        }


        /* Now calculate the donor vector as U(i)= X(1)+SCLAINGFACTOR(X(2) - X(3)) where
        * X(1), X(2), X(3) are the vectors in the currentGeneration and correspond to the value at the
        * indices i,j,k
        */

        /*Calculate F*(X(j)-X(k))*/
        for(pointer=0; pointer<DEvolution.DIMENSION; pointer++){
            temp[pointer] = (DEvolution.SCALINGFACTOR) *( (DEvolution.currentGeneration[j][pointer] - DEvolution.currentGeneration[k][pointer]) );
        }/*end of for that calculates X(2)-X(3)*/
```

Figure 4. Rand1.java

This class represents the mutation strategy *DE/rand/1*, which is the first of the mutation strategies proposed by Storn and Price [2]. It implements MutationStrategy which is an interface which must be implemented by every class representing any mutation strategy.

Major Responsibilities:

1. Mutation of the target vector.
2. Bounds checking of the mutated vector.

```
public class BinaryCrossover implements CrossOverStrategy{

    double[]    trial       =   new double[DEvolution.NP];
    double[]    survivor    =   new double[DEvolution.NP];
    int  jindex,DMpointer;
    double random;
    Random  rm  =   new Random();

    public void crossover(int target,Function F){

        /*jindex        = an integer randomly generated between 1 and the number of parents, which ensures
        *                   that the trial vector gets atleast some features from the donor
        *  DMpointer    = counter for iterating through the dimensions or features or genes etc
        */



        jindex  =   rm.nextInt(DEvolution.DIMENSION);

        for(DMpointer=0; DMpointer<DEvolution.DIMENSION ; DMpointer++){
            random  =   rm.nextDouble();
            if( (random <= DEvolution.CROSSOVERRATE) || (jindex==DMpointer) ){
                DEvolution.trial[DMpointer]=DEvolution.donor[DMpointer];
            }
            else{
                DEvolution.trial[DMpointer]=DEvolution.currentGeneration[target][DMpointer];
            }/*end of if*/

        }/*end of inner for*/

        /*call the selector which decides whether offspring(trial) of parent(target) should survive
        and updates the winner in global newGeneration space*/



        this.selector(target, DEvolution.currentGeneration[target],DEvolution.trial, F);

    }/*end of crossover*/
```

Figure 5. BinaryCrossover.java

This class represents the crossover strategy. It implements the interface CrossoverStrategy, which must be implemented by every class representing a crossover strategy.

Major Responsibilities:

1. Perform recombination of target and donor vectors.
2. Create the trial vector.
3. Select the better vector between target and trial vectors using the elitism selection strategy.

Due to lack of space only the important core classes were described in this section.

4. EXPERIMENTAL SETUP

4.1. IEEE CEC 2013 Test Suite

There are numerous real world problems that involve a large number of variables that need to be optimized. Various factors compound the problems faced in such large scale optimization problems. First, with the increase in the number of variables associated with a problem the search space grows exponentially. Second, the properties of the problem tend to change as the dimensionality of the problem increases. Third, computation of such large scale problems is expensive. The problem becomes even more complicated if the variables interact with each other and need to be optimized together instead of separately. This property of interaction between variables is generally known as *non-separability* in the optimization literature.

The best case in this scenario would occur when none of variables interact with each other, i.e. they are *fully separable*. The worst case would be if all of the variables interact, i.e. they are *non-separable.* But most of the real world problems lie in between those extremes [7]. For these problems only subsets of the total number of variables interact with each other. IEEE CEC 2013 Test Suite [7] is a set of benchmark functions that try to emulate the properties of real world large scale optimization problems to evaluate evolutionary algorithms. IEEE CEC Test suites have constantly evolved over time with the advances in the field of Large Scale Global Optimization commonly known as LSGO. In essence, it provides a framework on which to test and report the performance of EA.

All the problems listed in the Test Suite are minimization problems.

For the sake of completeness, the functions are described briefly. These functions are described in detail in [7]. To understand the benchmark functions, we need to be aware of the following terminology that is frequently used in the test suite.

$\boldsymbol{D}$ is Dimensionality of the problem

$\boldsymbol{O}$ is the shifted global minimum of the problem

$\boldsymbol{M_n}$ is the orthogonally rotated matrix obtained from the Gram-Schmidt ortho-normalization process.

$\boldsymbol{\Lambda^{\alpha}}$ is a diagonal matrix in D dimensions with ith diagonal value as $\alpha^{\frac{i-1}{2(D-1)}}$ for i =1,2…D

$T_{asy}^{\beta}: if\ \boldsymbol{x_i} > \boldsymbol{0}, \boldsymbol{x_i} = \boldsymbol{x_i}^{\boldsymbol{1+\beta\frac{i-1}{D-1}\sqrt{x_i}}}$

$T_{osz}: for\ \boldsymbol{x_i} = sign(\boldsymbol{x_i})\exp(\ddot{x} + 0.049(\sin(\boldsymbol{c_1}\ddot{x}_\iota) + \sin(\boldsymbol{c_1}\ddot{x}_\iota))), for\ i = 1\ and\ D$

$$where \qquad \ddot{x}_\iota = \begin{cases} \log\left(mod(\boldsymbol{x_i})\right)\ if\ \boldsymbol{x_i} \neq \boldsymbol{0} \\ 0 \qquad otherwise \end{cases}$$

$$sign(\boldsymbol{x_i}) = \begin{cases} -1 & if\ \boldsymbol{x_i} < 0 \\ 0 & if\ \boldsymbol{x_i} = \boldsymbol{0} \\ 1 & otherwise \end{cases}$$

$$\boldsymbol{c_1} = \begin{cases} 10 & if\ \boldsymbol{x_i} < 0 \\ 5.5 & otherwise \end{cases}$$

$$\boldsymbol{c_2} = \begin{cases} 7.9 & if\ \boldsymbol{x_i} < 0 \\ 3.1 & otherwise \end{cases}$$

Given these definitions, the functions are briefly described in the next section. Table 1 and subsequent figures of the benchmark functions are adapted from [7].

Table 1. IEEE CEC 2013 Function definitions and descriptions

	Function No.	**Function Name**	***fi*=fi(x*)(shifted global minimum)***
Unimodal Functions	1	Sphere Function	-1400
	2	Rotated High Conditioned Elliptic Function	-1300
	3	Rotated Bent Cigar Function	-1200
	4	Rotated Discus Function	-1100
	5	Different Powers Function	-1000
Multimodal Functions	6	Rotated Rosenbrock's Function	-900
	7	Rotated Schaffers F7 Function	-800
	8	Rotated Ackley's Function	-700
	9	Rotated Weierstrass Function	-600
	10	Rotated Griewank's Function	-500
	11	Rastrigin's Function	-400
	12	Rotated Rastrigin's Function	-300
	13	Non-Continuous Rotated Rastrigin's Function	-200
	14	Schwefel's Function	-100
	15	Rotated Schwefel's Function	100
	16	Rotated Katsuura Function	200
	17	Lunacek Bi_Rastrigin Function	300
	18	Rotated Lunacek Bi_Rastrigin Function	400
	19	Expanded Griewank's plus Rosenbrock's Function	500
	20	Expanded Scaffer's F6 Function	600

1. Sphere Function

$$f_1(x) = \sum_{i=1}^{D} z_i^2 + f_1^*, z = x - o$$

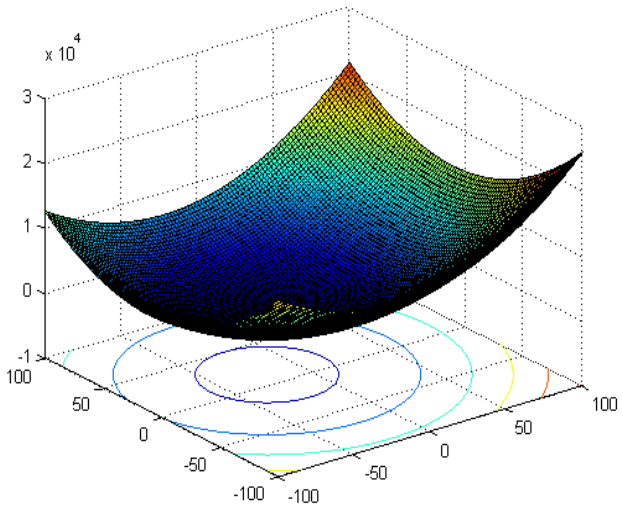

Figure 6. F1

Properties:

1. Unimodal
2. Separable

2. Rotated High Conditioned Elliptic Function

$$f_2(x) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} z_i^2 + f_2^*, \qquad z = T_{osz}(M_1(x - o))$$

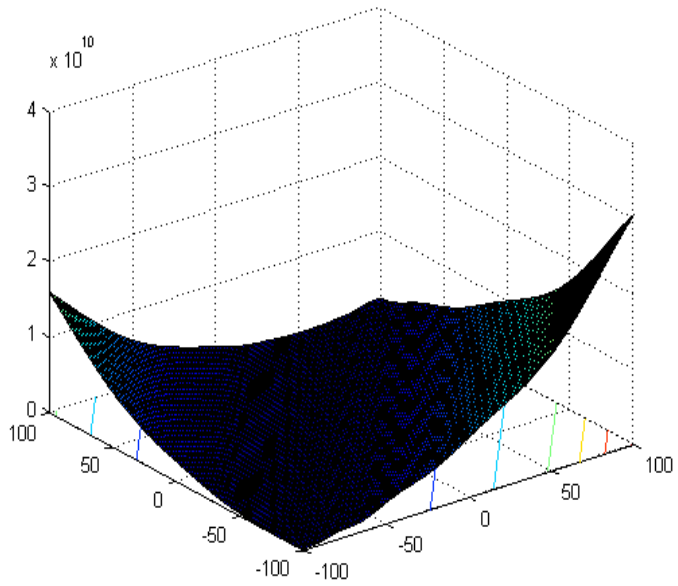

Figure 7. F2

Properties:

1. Unimodal
2. Non-Separable

3. Rotated Bent Cigar Function

$$f_3(x) = z_1^2 + 10^6 \sum_{i=2}^{D} z_i^2 + f_3^*,\ z = M_2(T_{asy}^{0.5}(M_1(x - o))$$

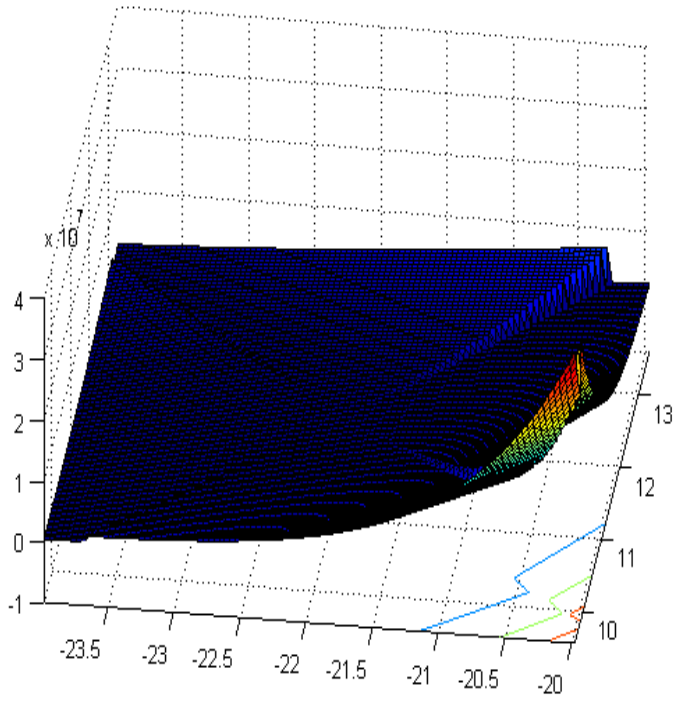

Figure 8. F4

Properties:

1. Unimodal
2. Non-Separable
3. Smooth but narrow ridge

4. Rotated Discus Function

$$f_3(x) = 10^6 z_1^2 + \sum_{i=2}^{D} z_i^2 + f_4^*, \qquad z = T_{osz}(M_1(x - o))$$

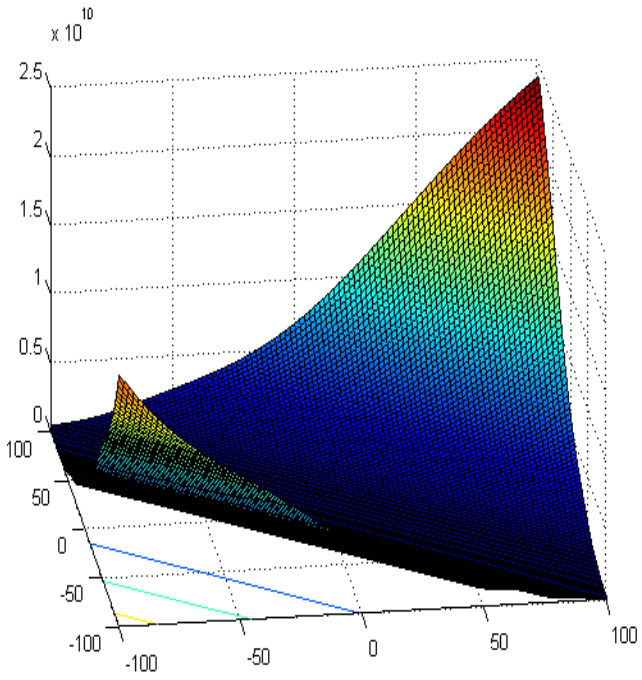

Figure 9. F4

Properties:

1. Unimodal
2. Separable
3. Asymmetrical
4. Smooth Local Irregularities

5. Different Powers Function

$$f_5(x) = \sqrt{\sum_{i=1}^{D} |z_i|^{2+4\frac{i-1}{D-1}}} + f_5^*, \qquad z = (x - o)$$

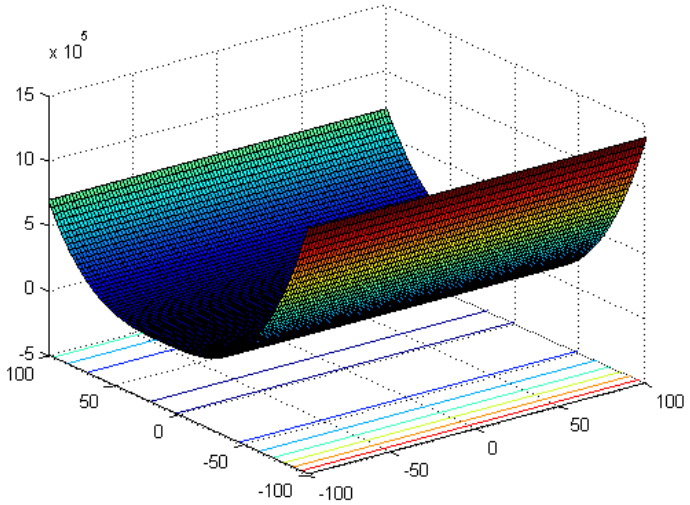

Figure 10. F5

Properties:

1. Unimodal
2. Separable

6. Rotated Rosenbrock's Function

$$f_5(x) = \sum_{i=1}^{D-1} \left(100\left(z_i^2 - z_{i+1}^2\right)^2\right) + (z_i - 1)^2 + f_6^*, \qquad z = M_1 \frac{2.048(x - o)}{100} + 1$$

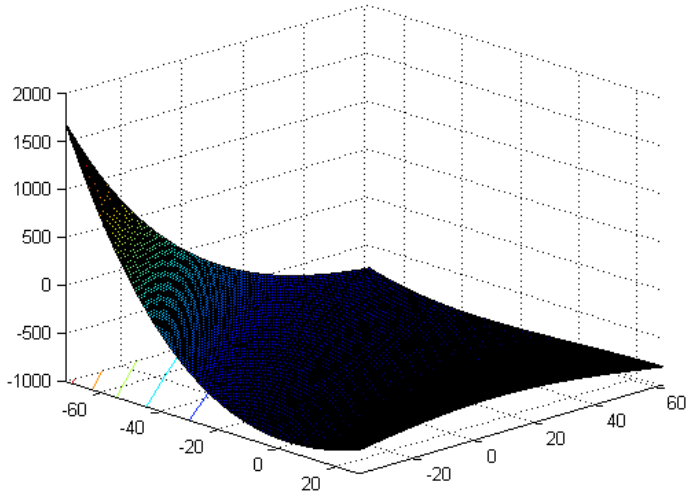

Figure 11. F6

Properties:

1. Multimodal
2. Non-Separable
3. Having a narrow valley from local optimum to global optimum

7. Rotated Schaffers F7 Function

$$f_7(x) = \left(\frac{1}{D-1}\sum_{i=1}^{D-1}\left(\sqrt{z_i} + \sqrt{z_i}sin^2(50{z_i}^{0.2})\right)\right)^2 + f_7^*$$

$$where\ z_i = \sqrt{y_i^2 + y_{i+1}^2}\ for\ i = 1, \ldots, D$$

$$and \quad y = \Lambda^{10} M_2 T_{asy}^{0.5}(M_1(x - o))$$

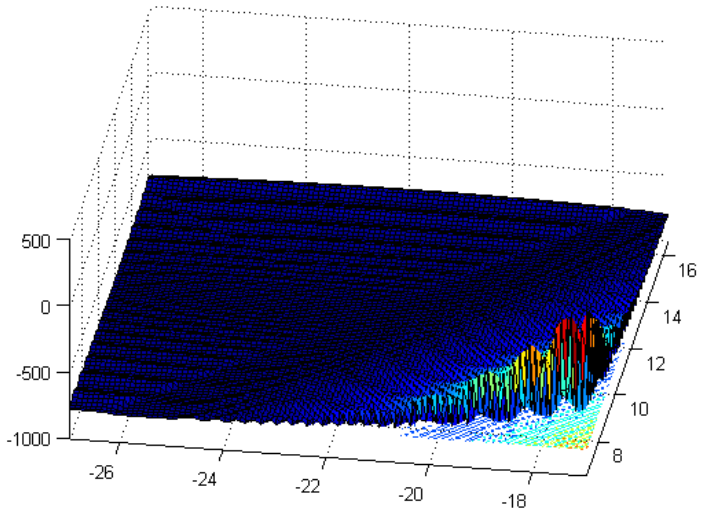

Figure 12. F7

Properties:

1. Multimodal
2. Non-Separable
3. Asymmetrical

8. Rotated Ackley's Function

$$f_8(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right) + 20 + e + f_8^*$$

$$where\ z = \Lambda^{10} M_2 T_{asy}^{0.5}(M_1(x - o))$$

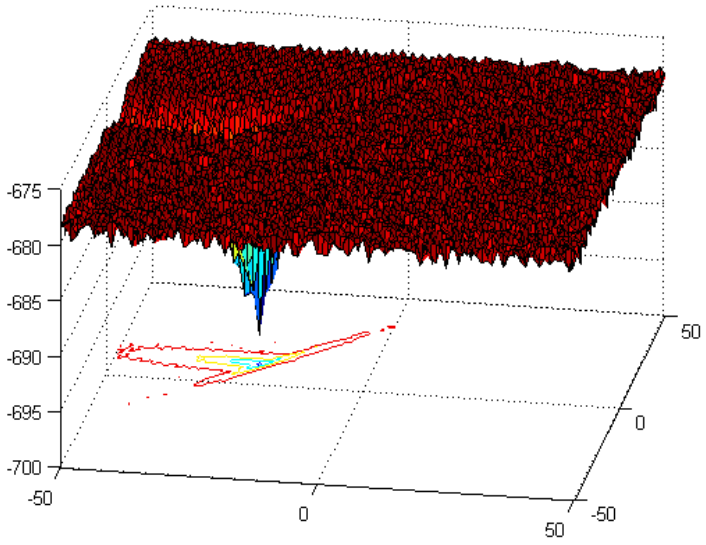

Figure 13. F8

Properties:

1. Multimodal
2. Non-separable

9. Asymmetrical Rotated Weierstrass Function

$$f_9(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{kmax}\left[a^k \cos\left(2\pi b^k(z_i + 0.5)\right)\right]\right) - D\sum_{k=0}^{kmax}\left[a^k \cos\left(2\pi b^k . 0.5\right)\right] + f_9^*$$

$$where\ a = 0.5, b = 3, kmax = 20$$

$$and\ z = \Lambda^{10} M_2 T_{asy}^{0.5}\left(M_1 \frac{0.5}{100}(x - o)\right)$$

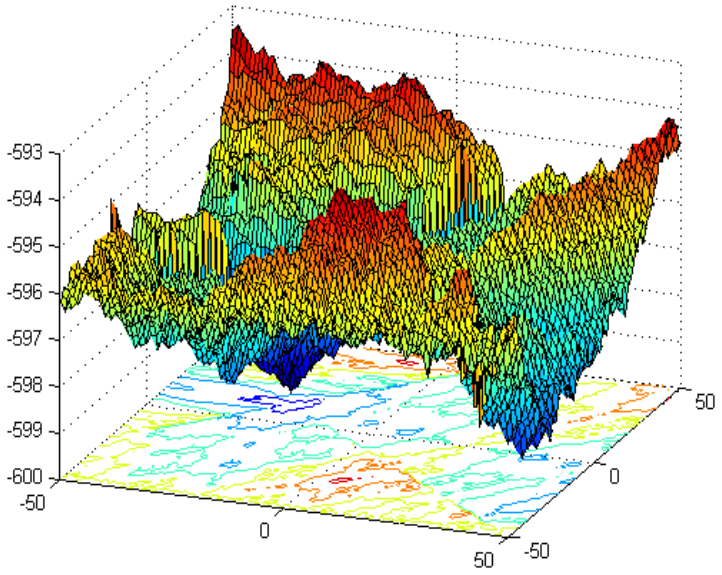

Figure 14. F9

Properties:

1. Multimodal
2. Non-Separable
3. Asymmetrical

10. Rotated Griewank's Function

$$f_{10}(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{10}^*$$

$$where\ z = \Lambda^{100} M_1 \frac{600(x - o)}{100}$$

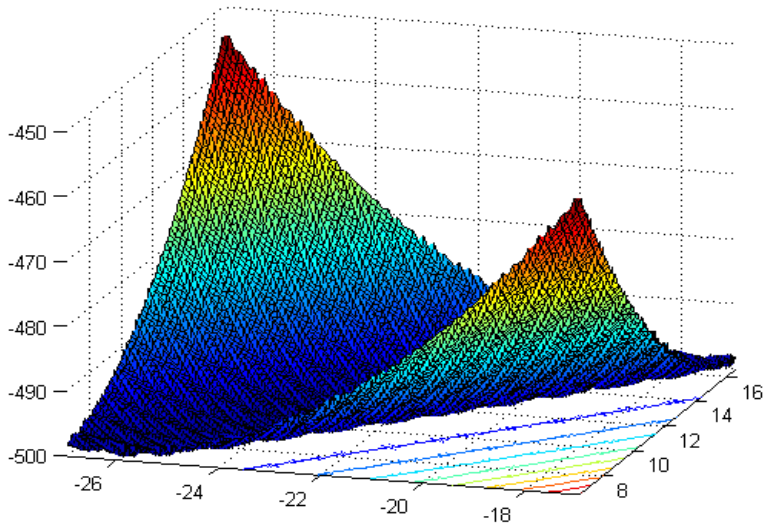

Figure 15. F10

Properties:

1. Multimodal
2. Non-separable

11. Rastrigin's Function

$$f_{11}(x) = \sum_{i=1}^{D}(z_i^2 - 10\cos(2\pi z_i) + 10) + f_{11}^*$$

$$z = \Lambda^{10} T_{asy}^{0.2}(T_{osz}\left(\frac{5.12(x-o)}{100}\right))$$

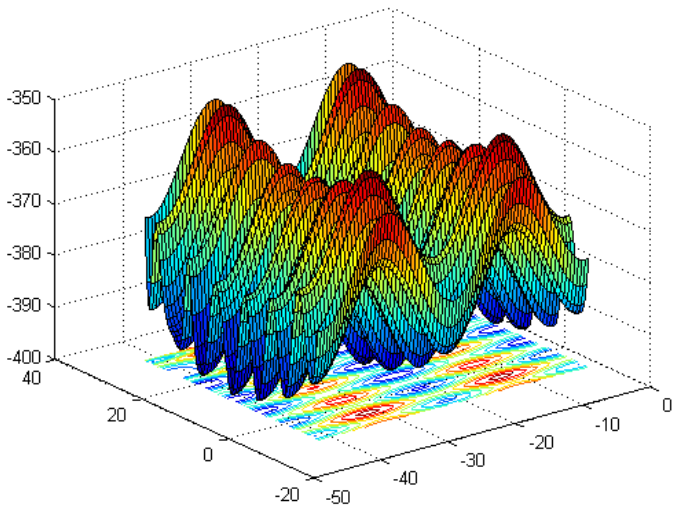

Figure 16. F11

Properties:

1. Multimodal
2. Non-separable
3. Huge number of local optima

12. Rotated Rastrigin's Function

$$f_{12}(x) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f_{12}^*$$

$$z = M_1 \Lambda^{10} M_2 T_{asy}^{0.2}(T_{osz}\left(M_1 \frac{5.12(x-o)}{100}\right))$$

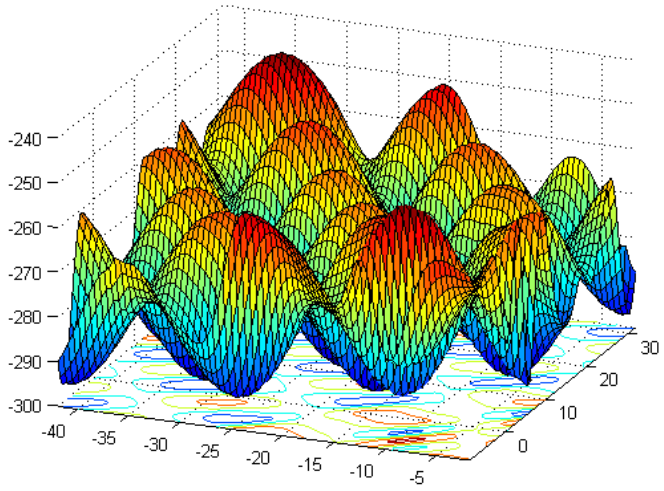

Figure 17. F12

Properties:

1. Multimodal
2. Non-separable
3. Asymmetrical
4. Huge number of local optima

13. Non-Continuous Rotated Rastrigin's Function

$$f_{13}(x) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f_{13}^*$$

$$\ddot{x} = M_i \frac{5.12(x-o)}{100}$$

$$y_i = \begin{cases} \ddot{x}_i & if\ |\ddot{x}_i| \le 0.5 \\ \dfrac{round(2\ddot{x}_i)}{2} & if\ |\ddot{x}_i| > 0.5 \end{cases} for\ i = 1, 2, \dots, D$$

$$z = M_1 \Lambda^{10} M_2 T_{asy}^{0.2}(T_{osz}(y))$$

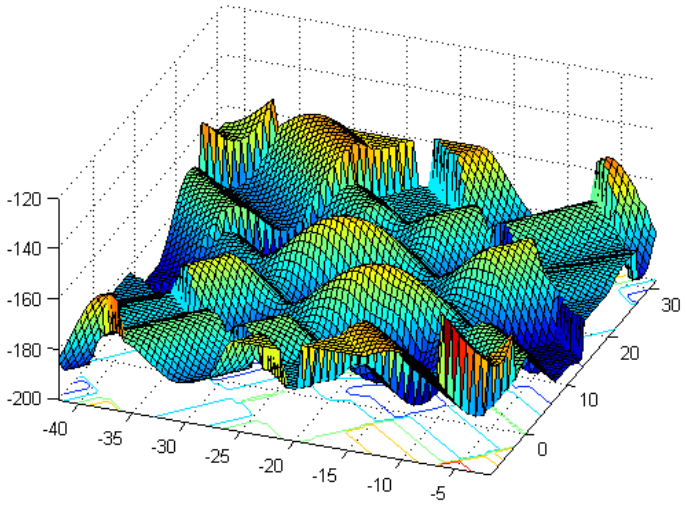

Figure 18. F13

Properties:

1. Multimodal
2. Asymmetrical
3. Huge number of local optima
4. Non-continuous

14. Schwefel's Function

$$f_{14} = 418.9829 \times D - \sum_{i=1}^{D} g(z_i) + f_{14}^{*}$$

$$z = \Lambda^{10}\left(\frac{1000(x - o_{-}}{100}\right) + 4.2096874227503e + 002$$

$$g(z_i) = \begin{cases} z_i \sin\left(|z_i|^{\frac{1}{2}}\right) & if\ |z_i| \le 500 \\ (500 - mod(z_i, 500))\sin(\sqrt{|500 - mod(z_i, 500)|}) - \frac{z_i - 500)^2}{10000D} & if\ z_i > 500 \\ (mod(z_i, 500) - 500)\sin\left(\sqrt{|mod(z_i, 500) - 500|}\right) - \frac{z_i + 500)^2}{10000D} & \text{if } z_i < -500 \end{cases}$$

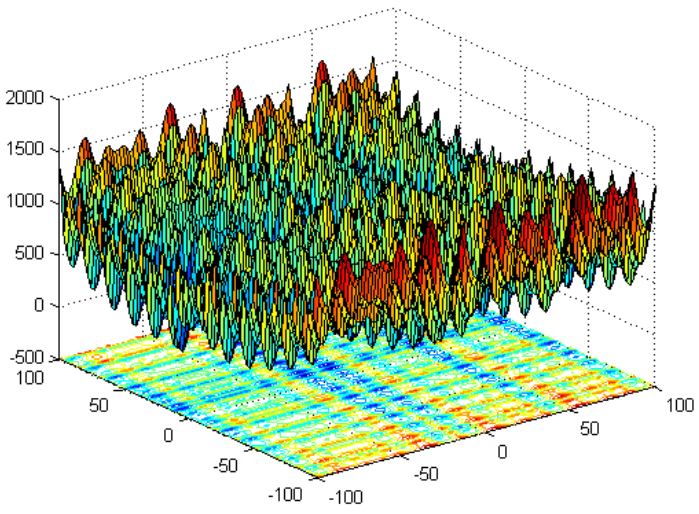

Figure 19. F14

Properties:

1. Multimodal
2. Non-separable
3. Huge number of local optima

15. Rotated Schwefel's Function

$$f_{15} = 418.9829 \times D - \sum_{i=1}^{D} g(z_i) + f_{15}^{*}$$

$$z = \Lambda^{10} M_1 \left(\frac{1000(x - o_{-})}{100} \right) + 4.2096874227503e + 002$$

$$g(z_i) = \begin{cases} z_i \sin\left(|z_i|^{\frac{1}{2}}\right) & if\ |z_i| \le 500 \\ \left(500 - mod(z_i, 500)\right)\sin\left(\sqrt{|500 - mod(z_i, 500)|}\right) - \frac{z_i - 500)^2}{10000D} & if\ z_i > 500 \\ (mod(z_i, 500) - 500)\sin\left(\sqrt{|mod(z_i, 500) - 500|}\right) - \frac{z_i + 500)^2}{10000D} & \text{if } z_i < -500 \end{cases}$$

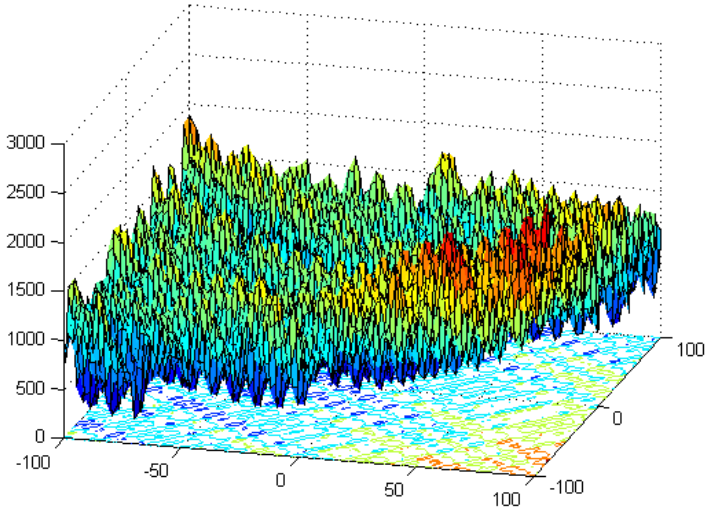

Figure 20. F15

Properties:

1. Multimodal
2. Asymmetrical
3. Non-separable

16. Rotated Katsuura Function

$$f_{16}(x) = \frac{10}{D^2}\prod_{i=1}^{D}\left(1 + i\sum_{j=1}^{32}\frac{2^j z_i - round(2^j z_i)}{2^j}\right)^{\frac{10}{D^{1.2}}} - \frac{10}{D^2} + f_{16}^*$$

$$z = M_2 \Lambda^{100}\left(M_1 \frac{5(x-o)}{100}\right))$$

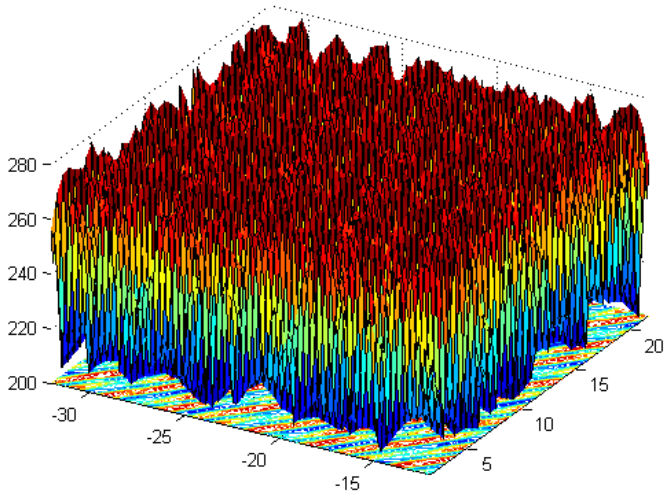

Figure 21. F16

Properties:

1. Multimodal
2. Non-separable
3. Asymmetrical

17. Lunacek Bi_Rastrigin Function

$$f_{17}(x) = \min(\sum_{i=1}^{D}(\ddot{x}_\iota - \mu_0)^2, dD + s\sum_{i=1}^{D}(\ddot{x}_\iota - \mu_1)^2 + 10(D - \sum_{i=1}^{D}\cos(2\pi z_\iota\ddot{)}) + f_{17}^*$$

$$\mu_0 = 2.5, \mu_1 = -\sqrt{\frac{\mu_0^2 - d}{s}}, s = 1 - \frac{1}{2\sqrt{D+20} - 8.2}, d = 1$$

$$y = \frac{10(x-o)}{100}, \ddot{x}_\iota = 2sign(x_i^*)y_i + \mu_0, for\ i = 1,2,\dots,D$$

$$z = \Lambda^{100}(\ddot{x} - \mu_0)$$

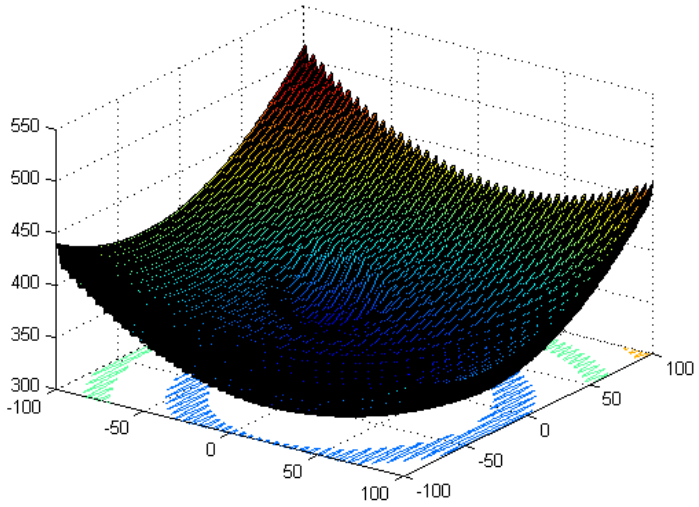

Figure 22. F17

Properties:

1. Multimodal
2. Asymmetrical

18. Rotated Lunacek Bi_Rastrigin Function

$$f_{18}(x) = \min(\sum_{i=1}^{D}(\ddot{x}_\iota - \mu_0)^2, dD + s\sum_{i=1}^{D}(\ddot{x}_\iota - \mu_1)^2 + 10(D - \sum_{i=1}^{D}\cos(2\pi z_\iota\ddot{)}) + f_{18}^*$$

$$\mu_0 = 2.5, \mu_1 = -\sqrt{\frac{\mu_0^2 - d}{s}}, s = 1 - \frac{1}{2\sqrt{D+20} - 8.2}, d = 1$$

$$y = \frac{10(x-o)}{100}, \ddot{x}_\iota = 2sign(y_i^*)y_i + \mu_0, for\ i = 1,2,\dots,D$$

$$z = M_2\Lambda^{100}(M_1(\ddot{x} - \mu_0))$$

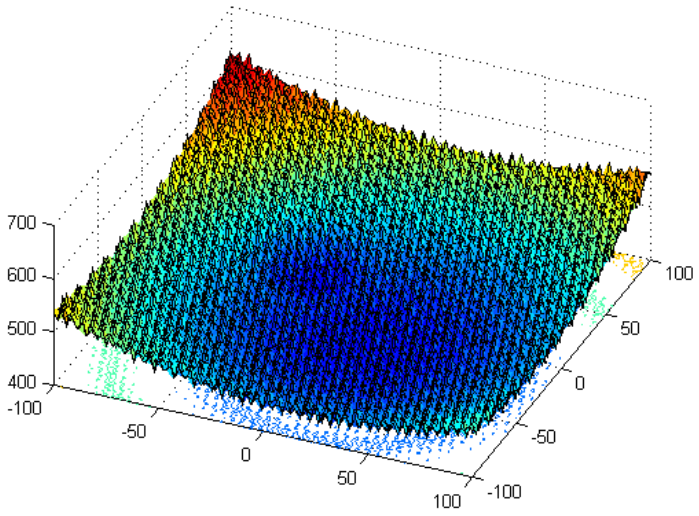

Figure 23. F18

Properties:

1. Multimodal
2. Non-separable
3. Asymmetrical

19. Expanded Griewank's plus Rosenbrock's Function

$$Basic\ Griewank'sFunction: \quad g_1(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$$

$$Baisc\ Rosenbrock'sFunction: \quad g_2(x) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2$$

$$f_{19}(x) = g_1\big(g_2(z_1, z_2)\big) + g_1\big(g_2(z_2, z_3)\big) + \cdots + g_1\big(g_2(z_{D-1}, z_D)\big) + g_1\big(g_2(z_D, z_1)\big) + f_{19}^*$$

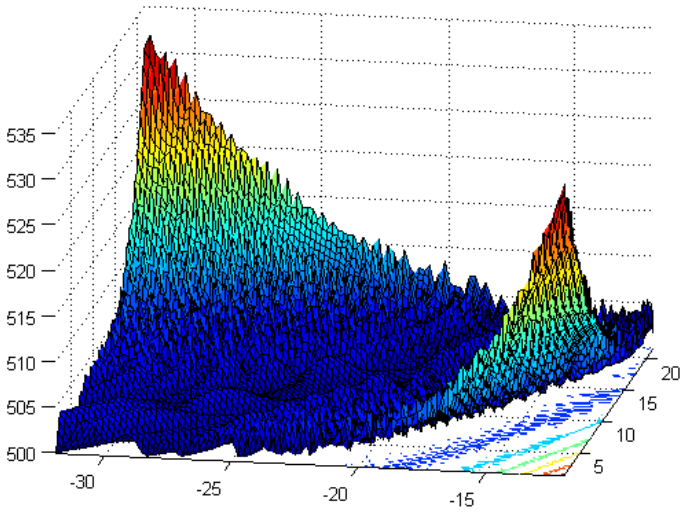

Figure 24. F19

Properties:

1. Multimodal
2. Asymmetrical

20. Expanded Scaffer's F6 Function

$$Scaffer's\ F6\ Function: g(x,y) = 0.5 + \frac{\sin^2\left(\sqrt{x^2+y^2}\right) - 0.5)}{\left(1 + 0.001(x^2+y^2)\right)^2}$$

$$f_{20}(x) = g(z_1, z_2) + g(z_2, z_3) + \cdots + g(z_{D-1}, z_D) + g(z_D, z_1) + f_{20}^*$$

$$z = M_2 T_{asy}^{0.5}\left(M_1(x-o)\right)$$

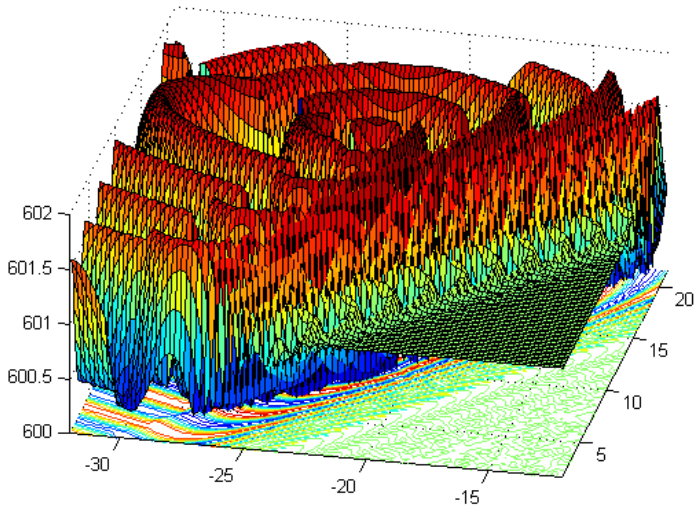

Figure 25. F20

Properties:

1. Multimodal
2. Non-separable
3. Asymmetrical

4.2. Experimental Settings

This paper attempts to evaluate the performance of the basic DE algorithm, i.e., *DE/rand/1/bin* on the first 20 benchmark functions specified by IEEE CEC 2013 Test Suite. The algorithm is executed for 10 different *NP* values *[20, 30, 50, 70, 80, 100, 150, 200, 250, 300]*, 3 *Cr* values *[0.1, 0.5, 0.9]* and 3 *F* values *[0.5, 0.8, 0.9]* at 3 different dimensions namely *10D*, *30D* and *50D*. Thus, for each function at given problem dimensionality, *10×3×3 = 90* evaluations are made. Let us denote a given evaluation set E_i as *[f_x, D, NP, F, Cr]* where:

f_x = *function to be optimized*

D = *Dimensionality of the problem*

NP = *number of parents/solutions*

F = *scaling factor*

Cr = *Crossover rate*

i = *evaluation set counter, 1 < i <90*

Every evaluation set is run 25 times and the best, worst, mean and standard deviation values are recorded for each evaluation set. For every function, the 90 Evaluation Sets $[E_1, E_2 \ldots E_{90}]$ form one evaluation matrix, *M*. One such evaluation matrix is shown below as a column matrix.

$$M = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ \vdots \\ E_{90} \end{bmatrix}$$

Since it would be highly cumbersome to report all the Evaluation Sets and Evaluation Matrices pertaining to every function, the best parameter setting results are reported for each function at a given dimensionality.

The test suite suggests two stopping criteria for the algorithm: 1) number of function evaluations reaches 10^4 times the problem dimensionality. This stopping criterion is significant as real world optimization tasks are computationally intensive. Thus, a predefined number of function evaluations serve as a cutoff parameter to the usually constrained computation budget. 2) The difference between the best value achieved so far and the global minimum (this difference is commonly known as Function Error Value, FEV) is smaller than 10^{-8}.

To gauge the performance of the algorithm at different combinations of control parameters (*NP*, *F*, *Cr*), the following measures are employed:

1. The best, worst and mean values of the FEVs achieved for every combination over 25 runs are compared at a given dimensionality.
2. The success rate of the combination. A combination run is said to have succeeded if it achieves FEV smaller than 10^{-8}.

5. RESULTS

Table 2 reports the performance of *DE/rand/1/bin* on benchmark functions at dimensionality *10D*. The best, worst and mean values shown are achieved by the best performing combination of control parameters among the 90 combinations, over 25 runs. For 10D problems, the algorithm is able to achieve the global minimum for 10 functions out of a total of 20 functions.

Table 2. Performance of *DE/rand/1/bin* at problem dimensionality 10

D	PM	F1	F2	F3	F4	F5
10 D	**Best**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	Worst	0.00e+00	0.00e+00	3.87e+00	0.00e+00	0.00e+00
	Mean	0.00e+00	0.00e+00	1.56e-01	0.00e+00	0.00e+00
	Std	0.00e+00	0.00e+00	2.85e-01	0.00e+00	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	**0.00e+00**	**0.00e+00**	**2.03e+01**	**0.00e+00**	**1.40e-02**
	Worst	0.00e+00	2.67e-04	2.08e+01	4.38e+00	1.18e-01
	Mean	0.00e+00	1.78e-05	2.03e+01	6.15e-01	4.92e-02
	Std	0.00e+00	4.15e-05	7.86e-02	6.15e-01	2.58e-02
		F11	**F12**	**F13**	**F14**	**F15**
	Best	**0.00e+00**	**2.98e-00**	**1.98e+00**	**0.00e+00**	**1.89e+01**
	Worst	0.00e+00	5.79e+01	2.95e+01	0.00e+00	1.30e+03
	Mean	0.00e+00	9.14e+00	1.25e+01	0.00e+00	4.24e+02
	Std	0.00e+00	6.62e+00	1.37e+00	0.00e+00	2.85e+02
		F16	**F17**	**F18**	**F19**	**F20**
	Best	**5.11e-01**	**1.86e+00**	**1.96e+01**	**4.07e-01**	**1.50e+00**
	Worst	1.89e+00	6.52e+01	8.71e+01	8.31e-01	4.12e+00
	Mean	1.02e+00	10.57e+00	3.50e+01	1.68e-01	2.96e+00
	Std	1.72e+01	3.97e+00	4.41e+00	8.86e-02	3.01e-01

Presented in figures from 22-41 are the *fitness versus generation* plots of DE algorithm at problem dimensionality *10D* for results of 20 functions.

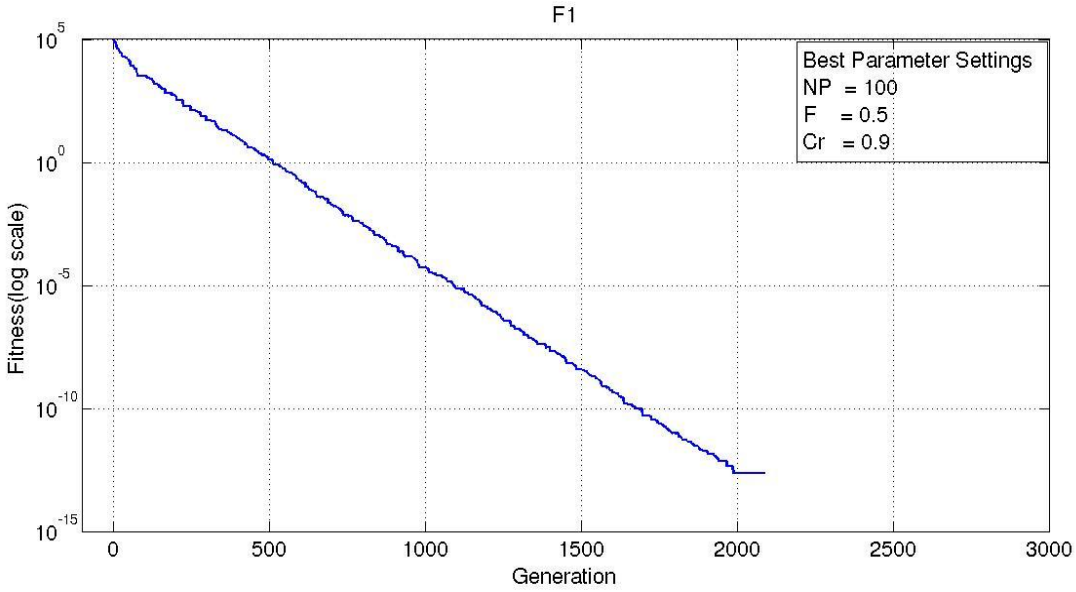

Figure 26. Average fitness curve of *DE/rand/1* for F1

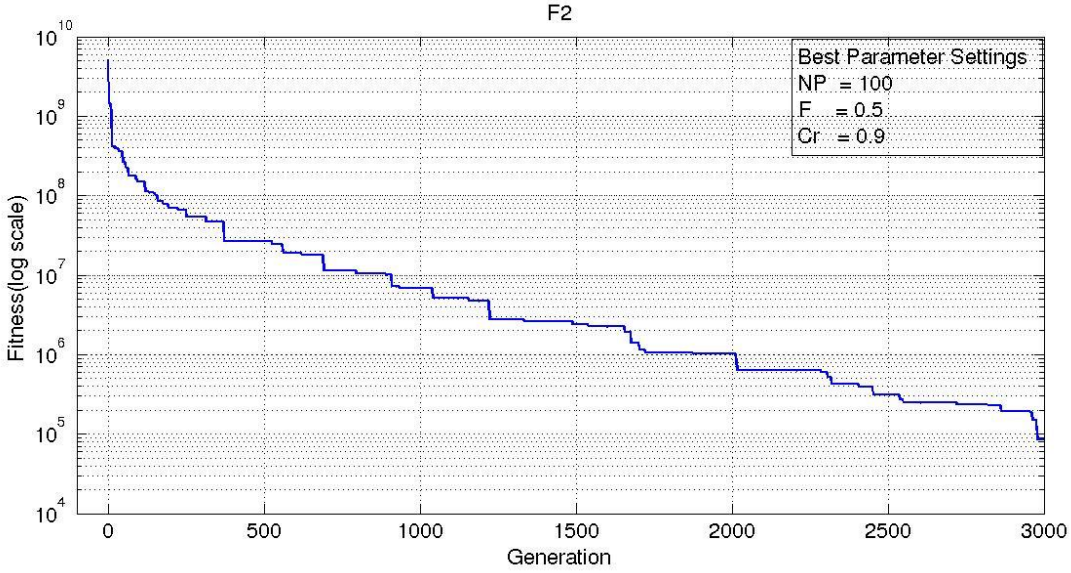

Figure 27. Average fitness curve of *DE/rand/1* for F2

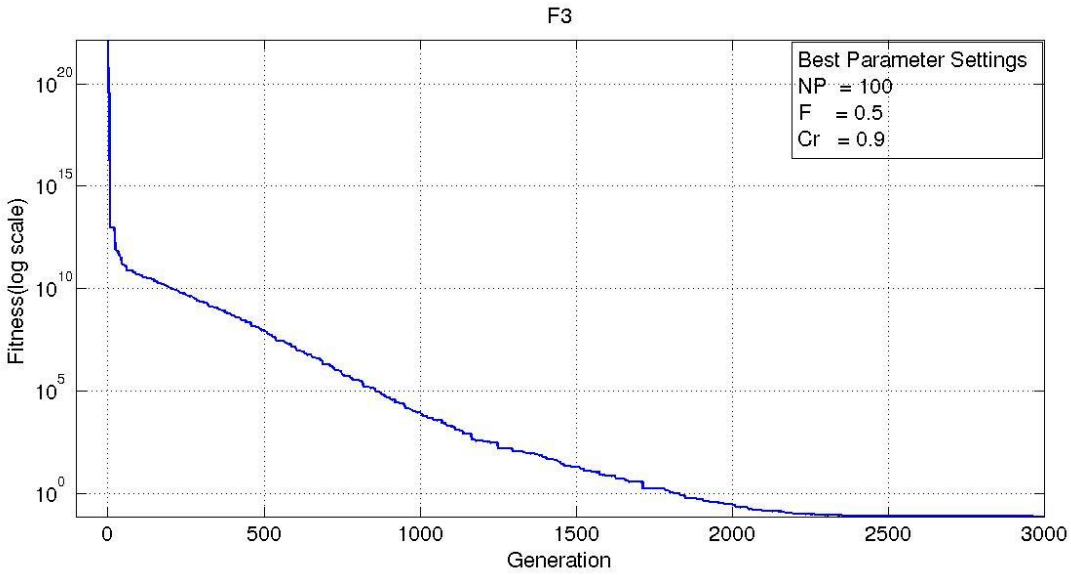

Figure 28. Average fitness curve of *DE/rand/1* for F3

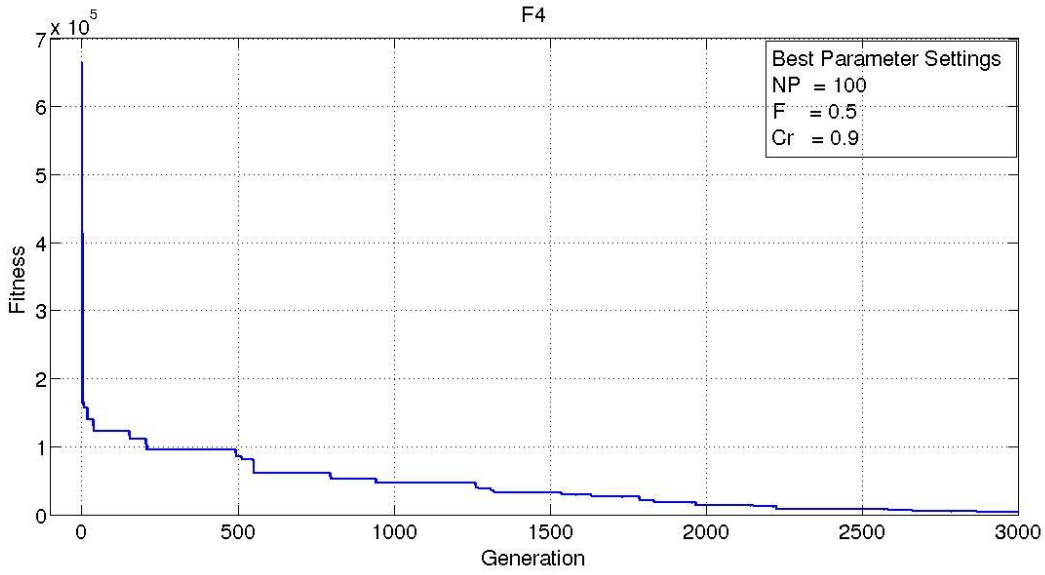

Figure 29. Average fitness curve of *DE/rand/1* for F4

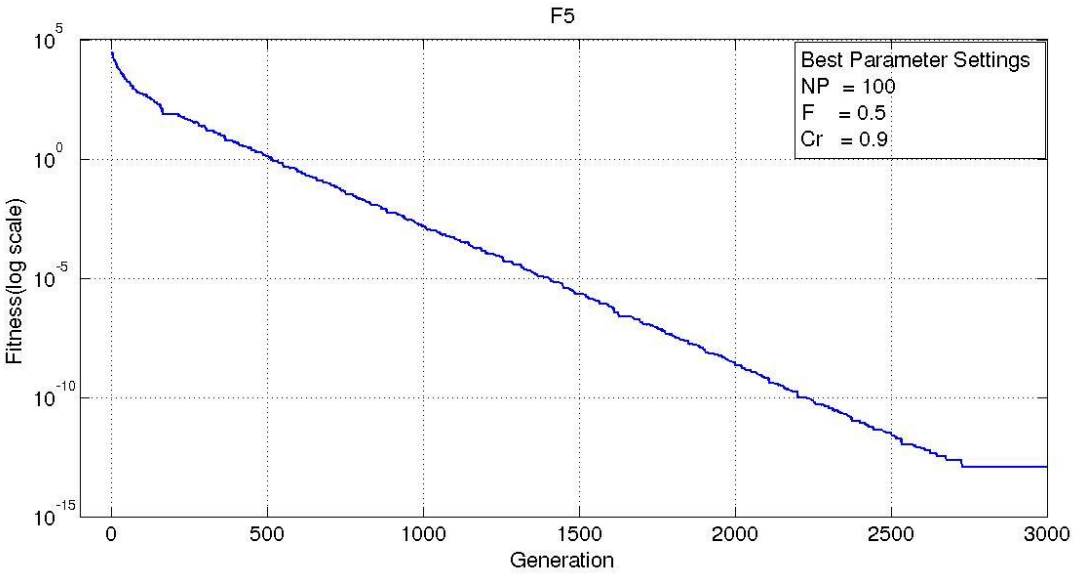

Figure 30. Average fitness curve of *DE/rand/1* for F5

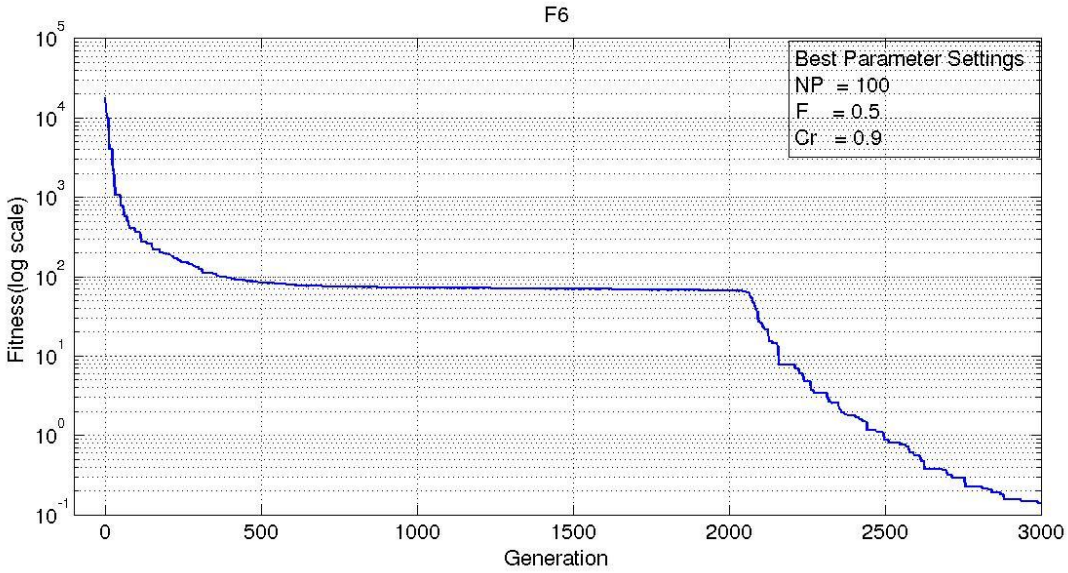

Figure 31. Average fitness curve of *DE/rand/1* for F6

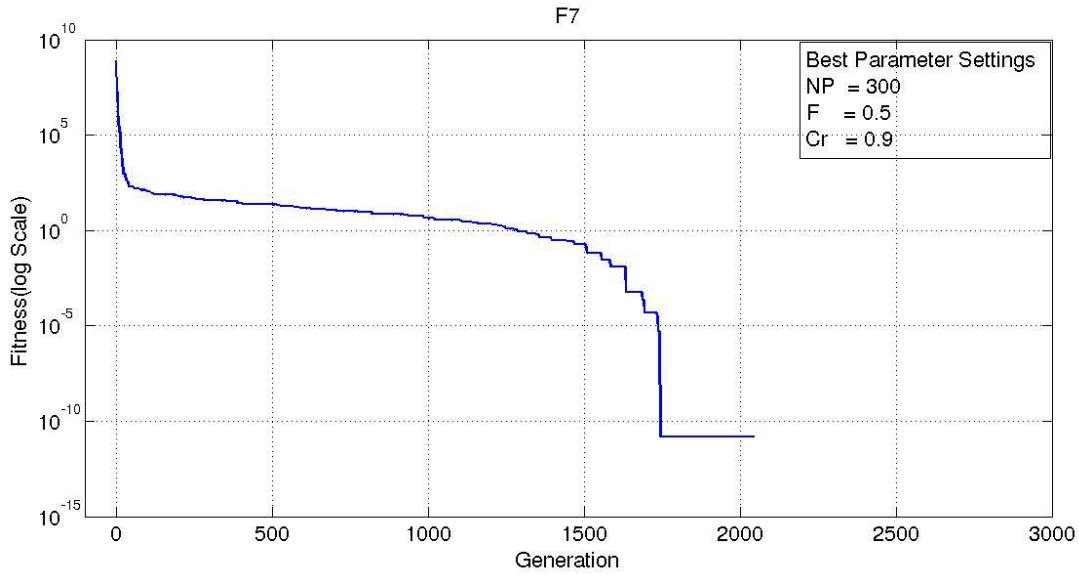

Figure 32. Average fitness curve of *DE/rand/1* for F7

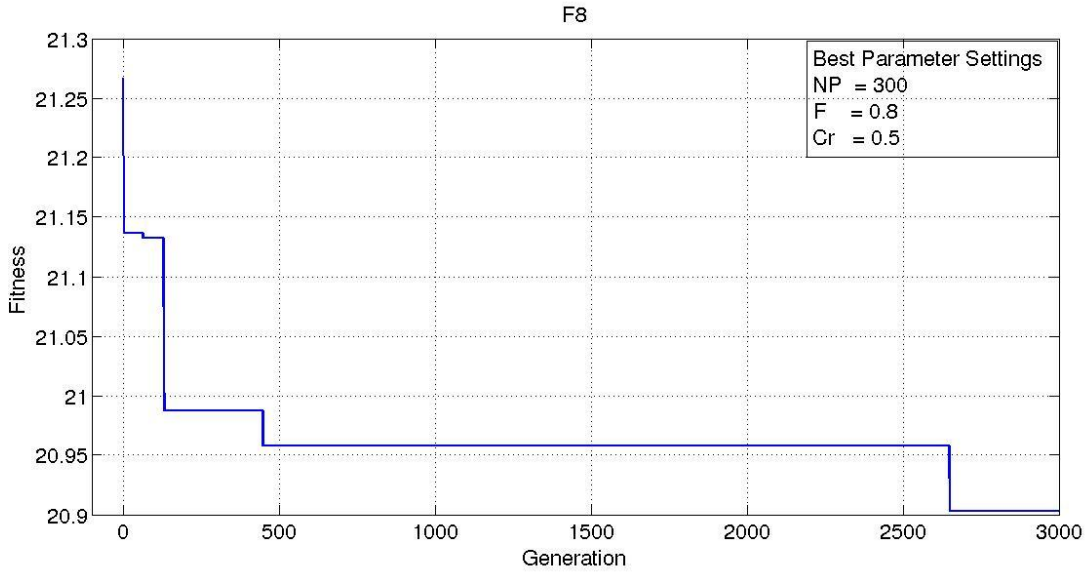

Figure 33. Average fitness curve of *DE/rand/1* for F8

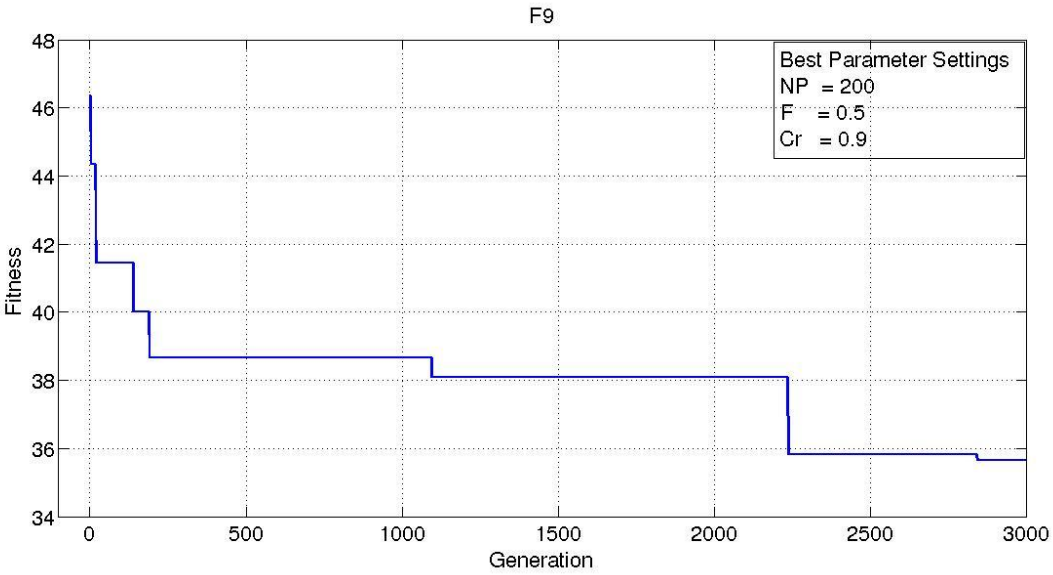

Figure 34. Average fitness curve of *DE/rand/1* for F9

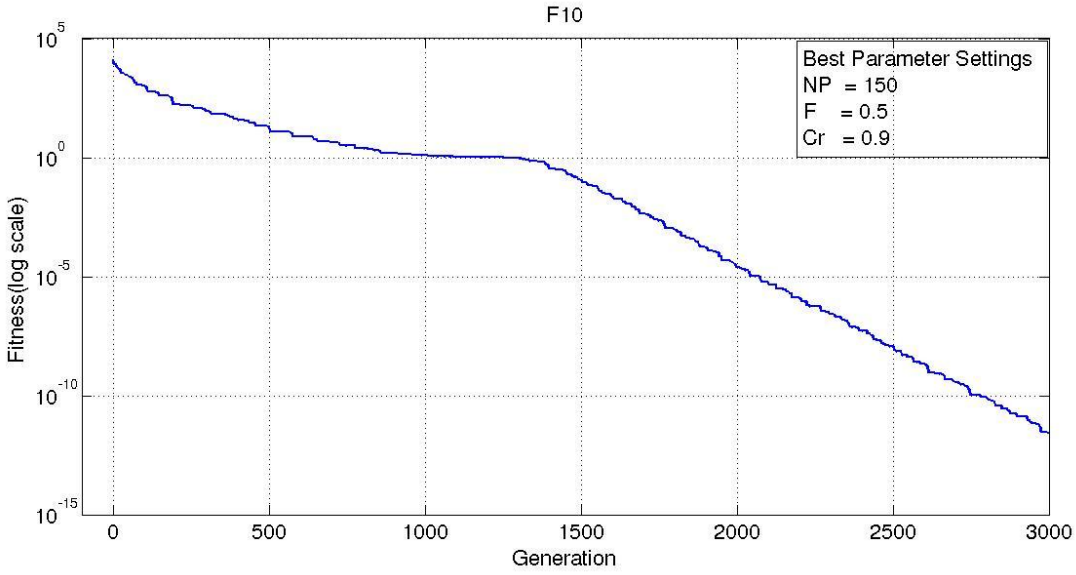

Figure 35. Average fitness curve of *DE/rand/1* for F10

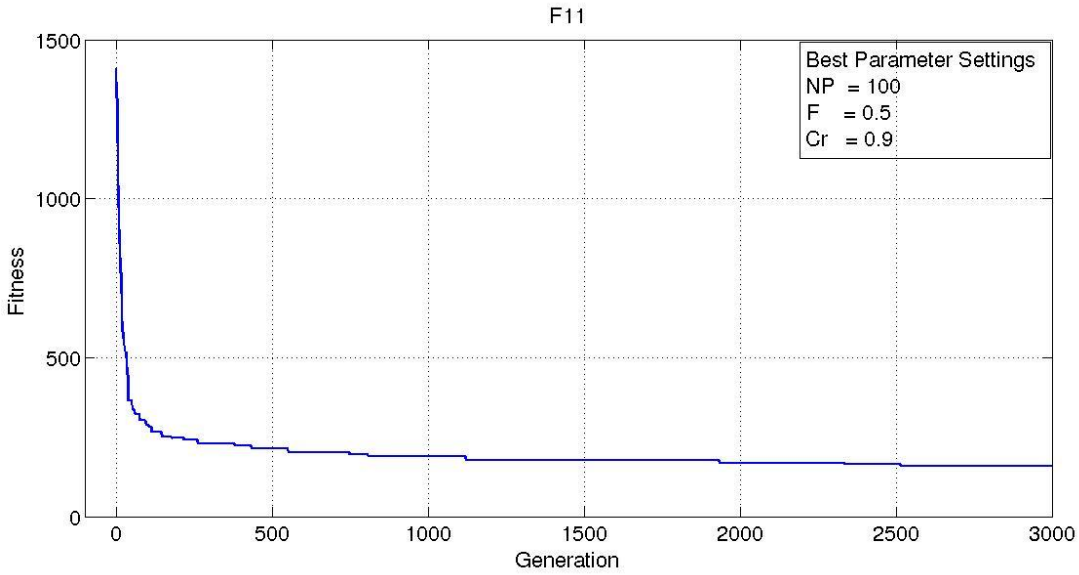

Figure 36. Average fitness curve of *DE/rand/1* for F11

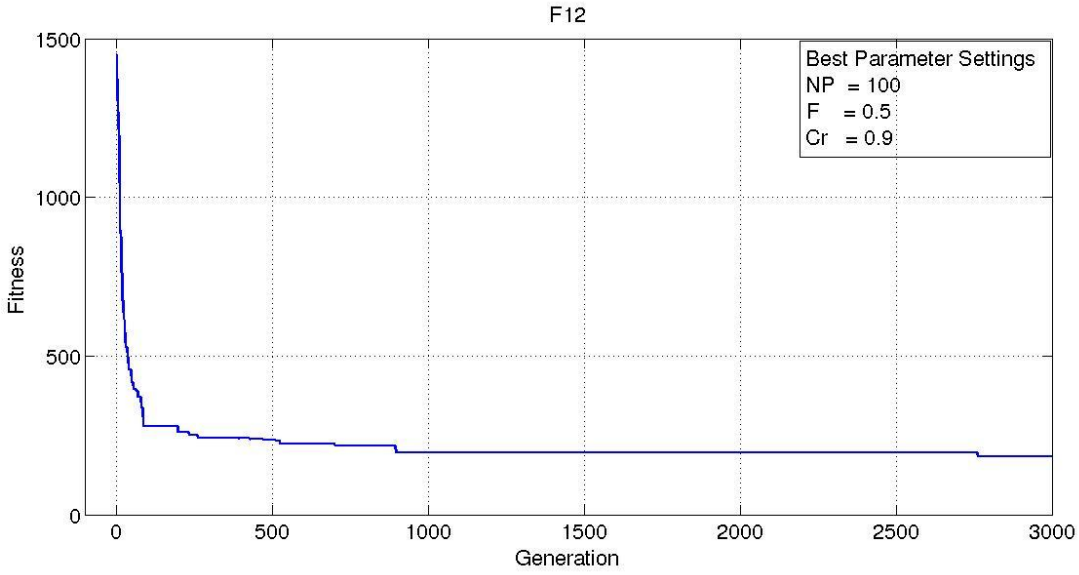

Figure 37. Average fitness curve of *DE/rand/1* for F12

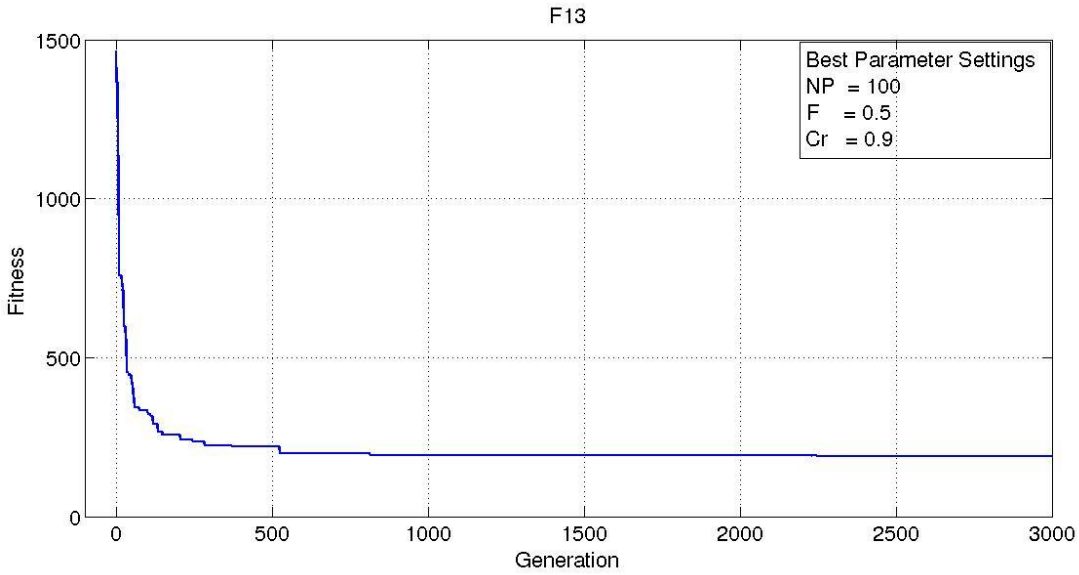

Figure 38. Average fitness curve of *DE/rand/1* for F13

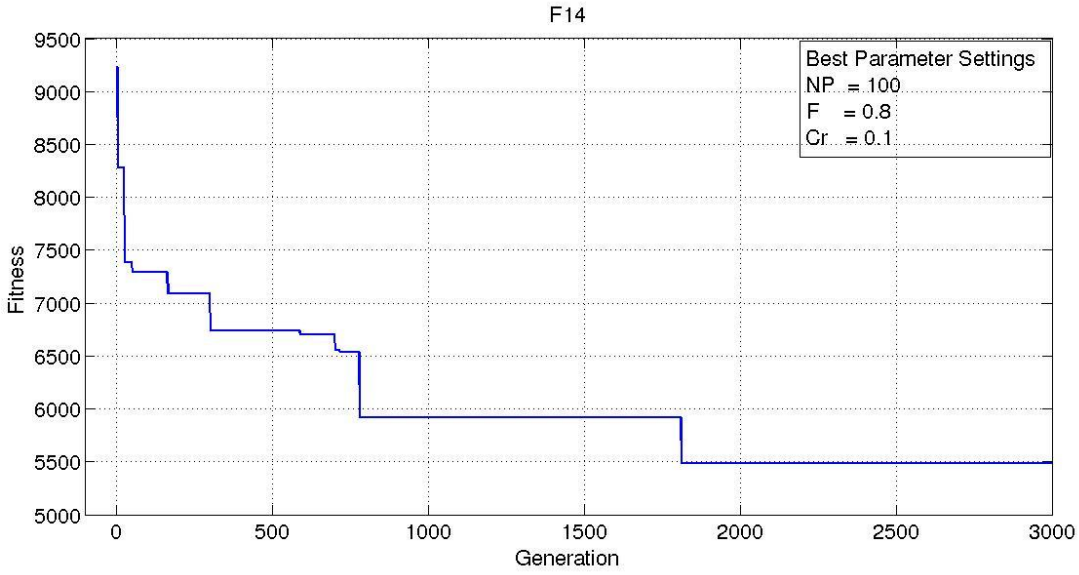

Figure 39. Average fitness curve of *DE/rand/1* for F14

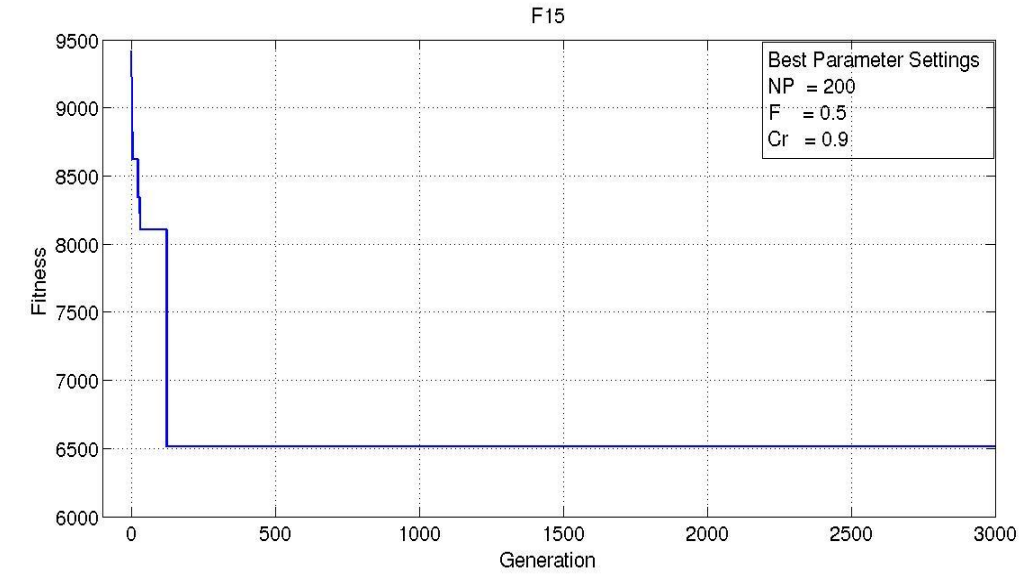

Figure 40. Average fitness curve of *DE/rand/1* for F15

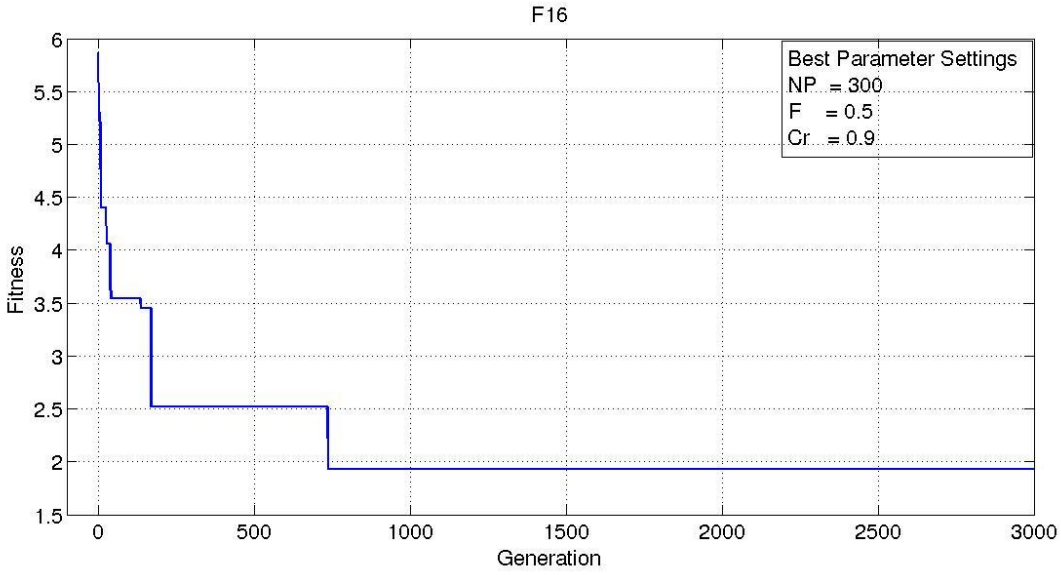

Figure 41. Average fitness curve of *DE/rand/1* for F16

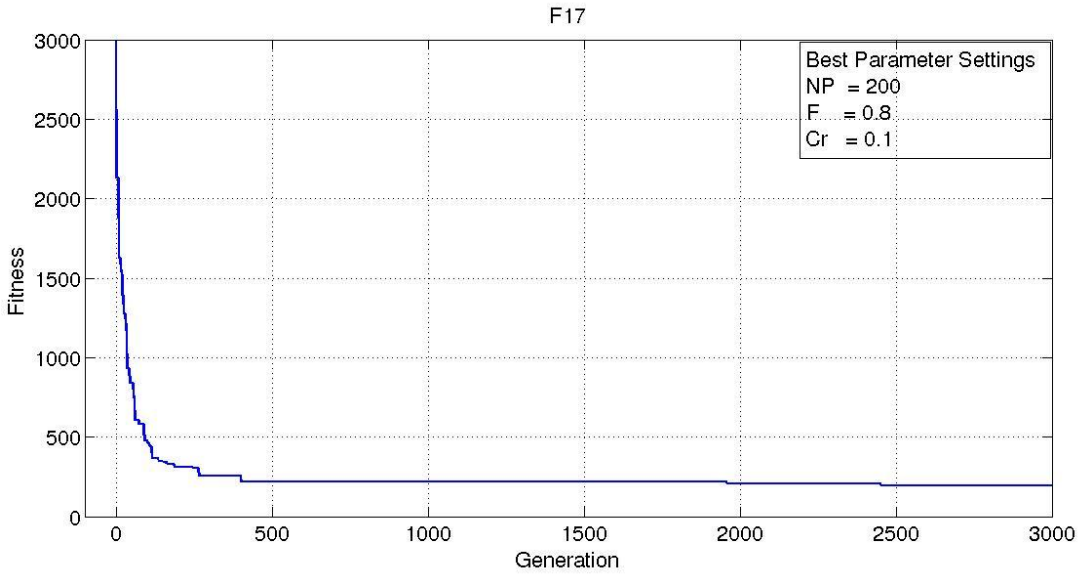

Figure 42. Average fitness curve of *DE/rand/1* for F17

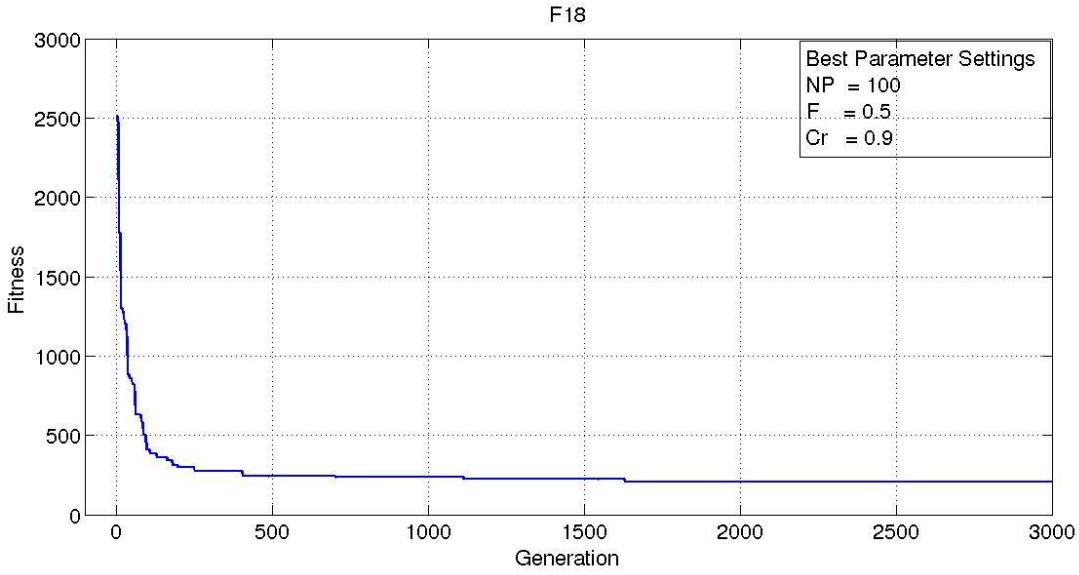

Figure 43. Average fitness curve of *DE/rand/1* for F18

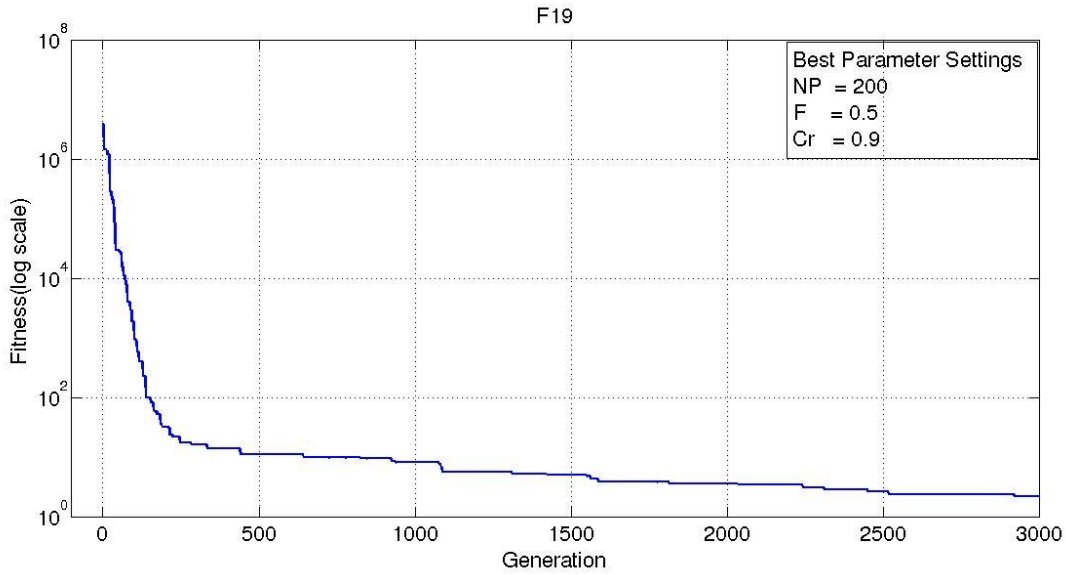

Figure 44. Average fitness curve of *DE/rand/1* for F19

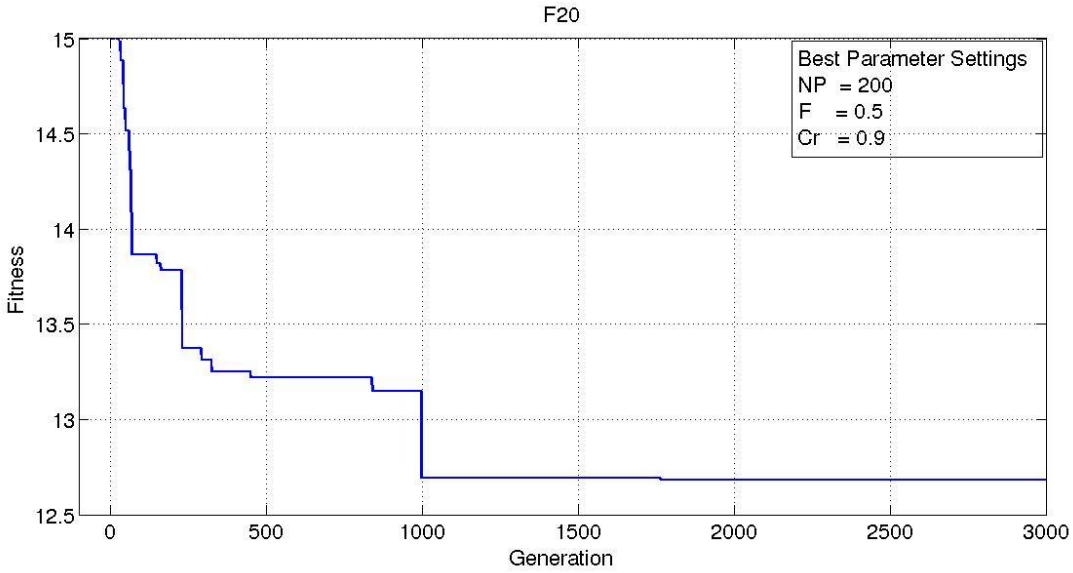

Figure 45. Average fitness curve of *DE/rand/1* for F20

Table 3. Performance of *DE/rand/1/bin* at problem dimensionality 30

D	PM	F1	F2	F3	F4	F5
30 D	**Best**	**0.00e+00**	**1.58e+04**	**0.00e+00**	**6.32e+02**	**0.00e+00**
	Worst	0.00e+00	1.84e+05	4.18 +00	1.69e+03	0.00e+00
	Mean	0.00e+00	8.49e+04	2.87e-01	1.59e+02	0.00e+00
	Std	0.00e+00	4.18e+04	1.34e+00	2.42e+02	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	**1.83e+00**	**0.00e+00**	**2.87e+01**	**4.21e+00**	**2.93e-08**
	Worst	2.14e+00	1.21e-06	2.12e+01	4.62e+01	1.23e-02
	Mean	1.32e+00	2.50e-04	2.19e+01	4.17e+00	1.02e-03
	Std	1.23e+00	3.56e-05	5.69e-02	2.14e-01	2.88e-03
		F11	**F12**	**F13**	**F14**	**F15**
	Best	**0.00e+00**	**2.34e+01**	**3.50e+01**	**1.25e-01**	**4.26e+03**
	Worst	0.00e+00	9.95e+01	2.02e+02	2.91e-01	6.41e+03
	Mean	0.00e+00	4.41e+01	1.02e+02	2.18e-01	5.59e+03
	Std	0.00e+00	1.88e+01	3.96e+01	4.41e-02	3.78e+02
		F16	**F17**	**F18**	**F19**	**F20**
	Best	**1.49e+00**	**3.04e+01**	**1.20e+02**	**1.19e+00**	**8.94e+00**
	Worst	2.75e+00	3.04e+01	2.19e+02	2.34e+00	1.40e+01
	Mean	2.19e+00	3.04e+01	1.89e+02	1.83e+00	1.15e+01
	Std	2.95e-01	0.00e-00	2.41e+01	2.63e-01	7.97e-01

Table 4. Performance of *DE/rand/1/bin* at problem dimensionality 50

D	PM	F1	F2	F3	F4	F5
50 D	**Best**	**0.00e+00**	**2.07e+05**	**9.01e-01**	**3.23e+02**	**0.00e+00**
	Worst	0.00e+00	9.74e+05	1.99e+04	3.66e+03	0.00e+00
	Mean	0.00e+00	4.70e+05	7.88e+02	1.38e+03	0.00e+00
	Std	0.00e+00	1.63e+05	2.83e+03	8.14e+02	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	**4.34e+01**	**6.73e-02**	**2.10e+01**	**1.87e+01**	**1.93e-07**
	Worst	4.34e+01	9.09e+00	2.12e+01	7.11e+01	4.44e-02
	Mean	4.34e+01	1.80e+00	2.11e+01	2.73e+01	1.70e-02
	Std	0.00e+00	1.94e+00	4.15e-02	7.55e+00	1.14e-02
		F11	**F12**	**F13**	**F14**	**F15**
	Best	**0.00e+00**	**6.18e+01**	**1.43e+02**	**4.00e-01**	**1.17e+04**
	Worst	0.00e+00	1.67e+02	4.41e+02	1.17e+01	1.38e+04
	Mean	0.00e+00	1.08e+02	2.91e+02	2.80e+00	1.29e+04
	Std	0.00e+00	2.59e+01	8.10e+01	1.94e+00	4.75e+02
		F16	**F17**	**F18**	**F19**	**F20**
	Best	**2.27e+00**	**5.08e+01**	**3.54e+02**	**3.40e+00**	**1.99e+01**
	Worst	3.72e+00	5.08e+01	4.18e+02	1.91e+01	2.35e+01
	Mean	3.18e+00	5.08e+01	3.99e+02	5.89e+00	2.13e+01
	Std	3.23e-01	2.09e-03	1.51e+01	2.73e+00	6.31e-01

Figures 2-21 show, for benchmark functions 1-20, the DE algorithm as it progresses through generations. These graphs can be conveniently called *fitness versus generation* plots. The plots show that the fitness of the population gets better with an increase in number of generations for all benchmark problems. This shows the intrinsic optimization capability of *DE/rand/1/bin.* For a few functions, the improvement in fitness is very marginal. For such functions the fitness is plotted on a log scale instead of linear scale.

Due to space constraints, the graphs for *30D* and *50D* are not listed here but they exhibit similar character as the *10D* graphs. Tables 1, 2 and 3 report the performance of *DE/rand/1/bin* on 10D, 30D and 50D benchmark problems, respectively. The values reported are the fitness values obtained for the best performing parameter settings from a set of 90 settings considered, each setting being evaluated 25 times. Best, Worst, Mean, Std are the **BEST**, **WORST**, **MEAN** and **STANDARD DEVIATION** values obtained at the stopping criteria.

A quick look at the results shows that the performance of the algorithm decreases as the dimensionality of the problem increases. This could be called *"the curse of dimensionality"* [8]. The curse of dimensionality emanates from the fact that the search space increases exponentially with an increase in dimension. Thus, more effective and efficient search strategies are required to deal with the problem. Also, with an increase in dimension, the number of functions for which the global minimum is reached, decreases. For example at dimension 10, the global minimum of 10 $(f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_9, f_{11})$ functions are reached. This number is reduced to three (f_1, f_5, f_{11}) at 30D and 50D. Apart from attributing the drop in performance with increased complexity at higher dimensions, some other factors may also be at play. For example, the Rosenbrock function is unimodal at 2 dimensions but becomes multimodal for higher dimensions [9].

As already stated, the other goal of this paper is to come up with a set of good control parameter settings, as a rule of thumb, that can lead to good performance on most of the benchmark functions. Since the test suite represents a diverse number of functional problems, the same parameter settings may as well be applied on real world problems while optimizing with DE.

Table 5 displays the best performing parameter settings for function **F1** for dimension *10D*. The best performing setting, i.e. the setting that leads to global optimal being achieved, is shown with a label "*". There would be times when the global minimum would not be achieved by any parameter setting. In such cases, the best parameter setting is denoted by "×". Due to paucity of space only a few of the tables representing the above performance could be shown.

Table 5. Best performing parameter settings for function F1 at 10D

F1-10D		*NP*									
F	*Cr*	20	30	50	70	80	100	150	200	250	300
0.5	0.1	*	*	*	*	*	*	*	*	*	*
	0.5	*	*	*	*	*	*	*	*	*	*
	0.9	*	*	*	*	*	*	*	*	*	*
0.8	0.1	*	*	*	*	*	*	*	*	*	*
	0.5	*	*	*	*	*	*	*	*	*	*
	0.9	*	*	*	*	*	*	*	*	*	*
0.9	0.1	*	*	*	*	*	*	*	*	*	*
	0.5	*	*	*	*	*	*	*	*	*	*
	0.9	*	*	*	*	*	*	*	*	*	*

Table 5 shows that DE is able to reach the global minimum for every parameter setting for function F1. Function F1 is the most basic and easy to optimize of all the benchmark functions. That is the reason why we see that every control parameter setting is effective in this case.

Table 6. Best performing parameter settings for function F5 at 10D

F5-10D		*NP*									
F	*Cr*	20	30	50	70	80	100	150	200	250	300
0.5	0.1										
	0.5										
	0.9				*	*	*	*	*		
0.8	0.1										
	0.5										
	0.9				*	*	*				
0.9	0.1										
	0.5										
	0.9										

Table 6 shows that for function F5, DE is able to reach the global minimum for only a few control parameter settings. This, in a way, confirms that the performance of DE is very sensitive to these control parameters. The global minimum is normally reached at medium values of *NP* (70 to150), medium values of *F* (0.5, 0.8) and high values of Cr (0.9).

For function F8, as Table 7 shows, DE is not able to reach the global minimum. The symbol × represents the minimum value reached by the algorithm. Again we can see that the minimum is reached at medium values of *NP* (150), medium values of *F* (0.5) and high values of Cr (0.9). Similar results are reported for *30D* and *50D* problems but due to lack of space they could not be shown here.

Table 7. Best performing parameter settings for function F8 at 10D

F8-10D		*NP*									
F	*Cr*	20	30	50	70	80	100	150	200	250	300
0.5	0.1										
	0.5										
	0.9							×			
0.8	0.1										
	0.5										
	0.9										
0.9	0.1										
	0.5										
	0.9										

It is clear from the results of Tables 5, 6, and 7 that the population size, *NP*, ranging from 80 to 150, at dimension *10D* with F=0.5 and $Cr \in [0.8, 0.9]$ lead to better performance for most of the benchmark functions.

6. CONCLUSIONS AND FUTURE WORK

As the intricacy of real world optimization problems increase so does the demand for highly efficient and robust optimization algorithms. This research work attempted to evaluate one such algorithm, Differential Evolution, on IEEE CEC 2013 Test Suite, which represents a wide variety of benchmark functions, and proposes a set of control parameter settings that may lead to good performance on most of the functions. The paper evaluated 90 different combinations of parameter settings: $NP \in$ [20, 30, 50, 70, 80, 100, 150, 200, 250, 300] $\times$ $Cr \in$ [0.1, 0.5, 0.9] $\times$ $F \in$ [0.5, 0.8, 0.9] for each function at three different dimensions $D \in$ [10,30,50]. The results indicate that medium values of *NP* [80, 100, 150], medium values of *F (0.5)* and higher values of *Cr* [0.8, 0.9] lead to better performance across dimensions. The two most significant Evaluation Sets, $E_{i=}[f_x, D, NP, F, Cr]$, that this paper proposes as a result of rigorous experimentation, that may be applied to most of the functions to achieve good results, are

$$\boldsymbol{E_1 = [f_x, 10\text{-}30\text{-}50, 100, 0.5, 0.9]}$$

$$\boldsymbol{E_2 = [f_x, 10\text{-}30\text{-}50, 150, 0.5, 0.9]}$$

These results can be used as a reference for future work on finding the best parameter setting for new benchmark functions. Moreover, to compare the significance of different parameter settings at different dimensions, standard statistical and parametric tests may be applied. Also, new and improved mutation techniques may be devised to improve the performance of DE on these benchmark problems. All in all, the potential for future research with DE is vast and challenging and may actually become a necessity given the rise in complexity of the real world problems.

7. REFERENCES

1. G. W. Greenwood, “Finding Solutions to NP Problems”, Published in proceedings CEC 2001, 815-822, 2001.
2. R. Storn and K. Price, “Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,” *J. Global Optimiz.*, vol. 11, pp. 341–359, 1997.
3. J. Liu and J. Lampinen, “On setting the control parameter of the differential evolution method,” in *Proc. 8th Int. Conf. Soft Computing (MENDEL 2002)*, 2002, pp. 11–18.
4. S. Das and P. Suganthan, “Differential Evolution: A Survey of the State-of-the-Art” in IEEE transactions on evolutionary computation, Vol. 15, NO. 1, February 2011.
5. K. Price, R. Storn, and J. Lampinen, “*Differential Evolution*—A Practical Approach to Global Optimization.” Berlin, Germany: Springer, 2005.
6. R. Gämperle, S. D. Müller, and P. Koumoutsakos, “A parameter study for differential evolution,” WSEAS NNA-FSFS-EC 2002. Interlaken, Switzerland, WSEAS, Feb. 11–15, 2002.
7. J. J. Liang, B.-Y. Qu, P. N. Suganthan, and A. G. Hern´andez-D´ıaz, “Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization,” Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang. Technological University, Singapore, Technical Report 201212, 2013.
8. R. E. Bellman, *Dynamic Programming*, ser. Dover Books on Mathematics. Princeton, NJ, USA: Princeton University Press / Mineola, NY, USA: Dover Publications, 1957/2003.
9. Y.-W. Shang and Y.-H. Qiu, “A Note on the Extended Rosenbrock Function,” *Evolutionary Computation*, vol. 14, no. 1, pp. 119–126, Spring 2006.