IMPLEMENTATION AND EVALUATION OF CMA-ES ALGORITHM

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

By

Srikanth Reddy Gagganapalli

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2015

Fargo, North Dakota

North Dakota State University
Graduate School

Title

IMPLEMENTATION AND EVALUATION OF CMA-ES ALGORITHM

By

Srikanth Reddy Gagganapalli

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:

Dr. Simone Ludwig
Chair

Dr. Saeed Salem

Dr. María de los Ángeles Alfonseca-Cubero

Approved:

11/03/2015	Dr. Brian M. Slator
Date	Department Chair

ABSTRACT

Over recent years, Evolutionary Algorithms have emerged as a practical approach to solve hard optimization problems in the fields of Science and Technology. 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 regarding differentiability or continuity of the objective function. The inspiration to learn evolutionary processes and emulate them on computer comes from varied directions, the most pertinent of which is the field of optimization. In most applications of EAs, computational complexity is a prohibiting factor. This computational complexity is due to number of fitness evaluations. This paper presents one such Evolutionary Algorithm known as Covariance Matrix Adaption Evolution Strategies (CMA ES) developed by Nikolaus Hansen, We implemented and evaluated its performance on benchmark problems aiming for least number of fitness evaluations and prove that the CMA-ES algorithm is efficient in solving optimization problems.

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. Saeed Salem from Computer Science Department and Dr. María de los Ángeles Alfonseca-Cubero from mathematics Department for being on my supervisory committee and providing their feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The constant strive for designing highly productive systems that respond in a quick and efficient way has made optimization a core concern for engineers. Optimization is a series of actions performed through continuous learning for achieving the best possible result for a problem under given circumstances. Thus, an optimization algorithm tries to find the best possible solution from all feasible solutions for a problem through a predefined approach. Unfortunately, many real-world problems are very challenging and require huge computational resources to solve, brute force search methods are useless in these cases [1]. They simply take too much time to find the optimal solution. Moreover, in these cases it is always appreciated to find a reasonably good enough solution (sub-optimal) saving a lot of computational time rather than trying to find the perfect solution.

There is no single method available for solving all optimization problems efficiently. There are a number of ways developed for solving the optimization problems, which include Mathematical programming methods, stochastic processes, and Statistical methods.

Stochastic algorithms are a family of algorithms, which try to optimize the fitness function through random search techniques. Evolutionary Algorithms are stochastic search methods that inherit the nature of natural biological evolution that allow a population of organisms to adapt to its environment for survival. The history of this field suggests there are different variants of Evolutionary Algorithms mainly Genetic Algorithms and Evolutionary Strategies. Evolution strategies were proposed in the 1960's by Rechenberg and further developed by Schwefel to optimize candidate solutions composed of real valued parameters [2]. The common underlying idea behind all these techniques is the 'Survival of the Fittest' to produce better and better approximations to an optimal solution. Given a function to be maximized/minimized, we initialize a population of random solution candidates (“individual”) and explore and improve the candidates fitness by a repeated process of selecting the candidates that seed the next generation by recombination and mutation until a solution of sufficient fitness is found [1].

Evolutionary Algorithms can be implemented with just one of Crossover or Mutation operator. However, the research on nature-inspired computation learning suggest the use of both Crossover and Mutation operators for a faster convergence to a global optimum [12].

In an attempt to generate better individuals, two or more individuals with different but desired features are combined to produce a new generation by a process called **Crossover** anticipating individuals with better fitness than the individual’s in the current generation. Crossover of the population is achieved using 1) Discrete Recombination**,** where the offspring solutions inherit its components from one or other of two randomly selected parents. 2) Intermediate Recombination, where the offspring solutions inherit its components, which are weighted average of components, from two or more selected parents.

Mutation applied to Evolutionary computation, is achieved my adding normally distributed random values to each component of an individual at a generation. Evolution Strategies algorithms monitor and update the mutation strength dynamically at each generation (typically standard deviation of normal distribution) as the individuals approach closer to the global optima, thus, mutation is a primary operator in evolutionary strategies.

Once the offspring is generated by the process of Crossover and Mutation, the **Selection** of candidate to parent next generation is based on fitness ranking of individuals in offspring and one of the selection strategy. $(\mu + \lambda)$ - ES [5], where the mutants and parents compete with each other, (μ , λ) – ES [5], where just mutants compete with each other to parent the next generation.

The parent and mutant population can be as low as 1, which results in $(1+\lambda)$-ES or $(1,\lambda)$-ES, (1,1)-ES and (1+1)-ES [3][4].

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) developed by Nikolaus Hansen is an evolutionary algorithm for difficult non-linear non-convex black-box optimization problems in continuous domains [3]. The CMA-ES is considered as state-of-the-art in evolutionary computation and has been adopted as one of the standard tools for continuous optimization in many (probably hundreds of) research labs and industrial environments around the world that is proved to converge at optimal solution in very few generations, thus, decreasing the time complexity [8]. Some of the key features of CMA-ES are:

- CMA-ES is feasible on non-separable and/or badly conditioned problems since it is a second order approach estimating a positive definite matrix within an iterative procedure [7].
- In contrast to quasi-Newton methods, the CMA-ES does not use or approximate gradients and does not even presume or require their existence. This makes the method feasible on non-smooth and even non-continuous problems, as well as on multimodal and/or noisy problems [7].
- It turns out to be a particularly reliable and highly competitive evolutionary algorithm for local optimization and, surprising at first sight, also for global optimization.
- The CMA-ES does not require tedious parameter tuning for its application. In fact, the choice of strategy internal parameters is completely automated.

This paper is composed of six chapters, which are as follows: Chapter 1 introduces the basic ES and CMA-ES algorithms. Chapter 2 describes CMA-ES in detail. Chapter 3 presents the implementation details of the algorithm with a snapshot of pseudo code and Java implementation. Chapter 4 presents the IEEE CEC’ 2013 Test Suite and lists its first 20 benchmark functions for the Optimization Algorithms. The experimental setup to test the CMA-ES 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 discusses the results obtained with other optimization algorithms.

2. COVERIANCE MATRIX ADAPTION - EVOLUTION STRATEGIES (CMA-ES)

The CMA-ES algorithm was first proposed by Nikolaus Hansen in 2001. It is a bio-inspired optimization algorithm. CMA-ES relies on a distribution model of a candidate population (Parameterized Multivariate Normal Distribution) in order to explore the design space [1]. It is based on selection and adaption strategy of the sample population while preserving and modifying the strategy parameters, the convergence property of previous generations (Covariance Matrix), and utilizing the knowledge in generating the next generation population. In each generation the parent for the next generation is calculated with a weighted average of λ selected candidates from μ offsprings generated at that generation using a (λ, μ)-selection. The next generation population is generated by sampling a multi-variate normal distribution of the Covariance Matrix with the Variance at the generation g over the mean of the generation $M \quad N\left(M^{(g)}, (\sigma^{(g)})^2\, C^{(g)}\right)$ [3]. The step size $\sigma^{(g)}$ determines the overall variance of the mutation at generation g. The variable property of step size σ at each generation plays a vital role in controlling the premature convergence and close convergence to global optima.

The CMA-ES works through a cycle of stages represented in Figure 1.

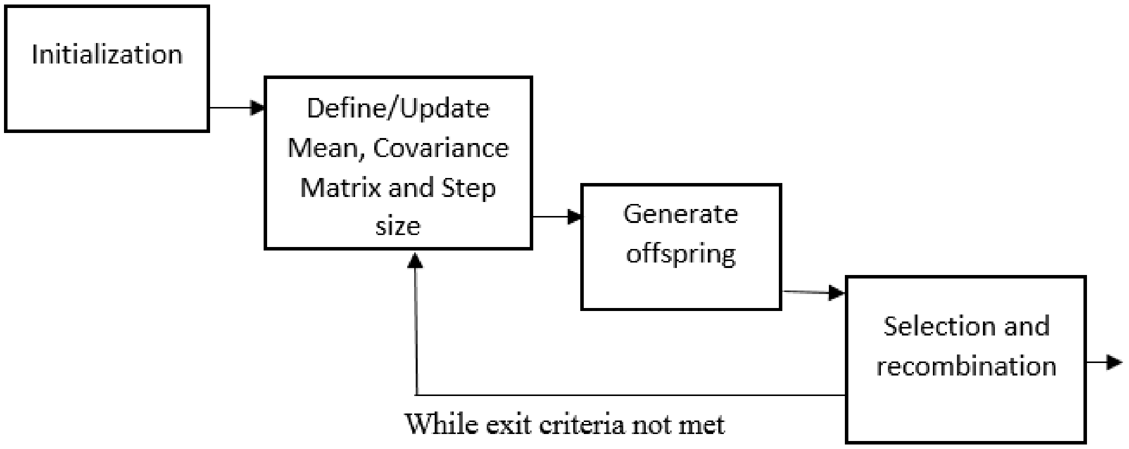

Initialization
Define/Update Mean, Covariance Matrix and Step size
Generate offspring
Selection and recombination
While exit criteria not met

Figure 1. Evolutionary strategies life cycle.

CMA-ES Search for the Global optima in D dimensions begins with generation of μ D-dimensional real valued members by sampling a multivariate Normal distribution around the Mean M at any generation. In general, the equation can be written as:

$$x_k^{(g+1)} \sim N\left(M^{(g)}, (\sigma^{(g)})^2\, C^{(g)}\right) \quad \text{for k = 1,…,}\lambda \tag{1}$$

Where $x_k^{(g+1)}$ represent the k^{th} Sample member generated at generation (g+1). Each member of the population can be broadly termed as a vector and can be represented as:

$$X_i = [x_1, x_2, x_3, \dots., x_D] \qquad \text{For i = 1,2,3,…, μ} \tag{2}$$

Thus from Equation (1), in order to pass to the next generation (g+2), the parameters $M^{(g+1)}$, $C^{(g+1)}$, $\sigma^{(g+1)}$ needs to be calculated. Each of these parameters represents a step in the algorithm and are discussed in next three sections.

2.1. Selection and Recombination

The crossover nature of the Evolutionary process is achieved by calculating the Mean Vector for every generation and then mutating the mean vector to generate the offspring. The mean vector M(g+1) for generation g is the weighted average of the μ selected best individuals in the order of the fitness ranking on the objective function from sample space $x_k^{(g+1)}$ for k = 0,1,2,…, μ. The weighted vector can be an equal vector or linear vector. With equal weighted vector, all the selected samples will have equal portion in the resultant mean (W_i = 1/ λ). With the Linear Weight vector configurationW_i, the fittest point has higher portion of genes than the point with lower fitness.

$$W_i = \log(\mu + 1) - \log(i + 1), \sum w_i = 1 \text{ , } W_1 > W_2 > W_3 > \cdots W_\lambda > 0 \text{ for } i = 1,2\ldots \lambda. \qquad (3)$$

2.2. Covariance Matrix Adaption

The Covariance Matrix Adaption determines the varying mutation for the child population in the evolutionary process. The offspring at a generation are sampled according to multivariate normal distribution in R^n, while Recombination described in Section 2.1 amounts to selecting a new mean at generation g+1, the mutation amounts to the sampling of the normal distribution of Covariance Matrix multiplied by Step Size around Mean. The dependencies between the variables in the distribution are represented by the Covariance Matrix. The Covariance Matrix Adaption is a method to update the covariance Matrix of this distribution, which amounts to learning a second order model of the underlying objective function.

The mean of the distribution is updated in such a way that the likelihood of the previously successful candidate solution is maximized, and the covariance Matrix is updated in such a way that the likelihood of previously successful search steps is increased.

$$\boldsymbol{C}^{(g+1)} = \left(\boldsymbol{1} - \boldsymbol{c_1} - \boldsymbol{c}_{\mu}\right)\boldsymbol{C}^{\boldsymbol{g}} + \boldsymbol{c}_1 \boldsymbol{P}_{\boldsymbol{c}}^{(\boldsymbol{g}+\boldsymbol{1})} \boldsymbol{P}_{\boldsymbol{c}}^{(\boldsymbol{g}+\boldsymbol{1})^{\boldsymbol{T}}} + \boldsymbol{c}_{\mu}$$

$$\times \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\mu} \boldsymbol{w}_{\boldsymbol{i}} \left(\frac{\boldsymbol{X}_{\boldsymbol{1}:\lambda}^{(\boldsymbol{g}+\boldsymbol{1})} - \boldsymbol{M}^{\boldsymbol{g}}}{\sigma^{(g)}}\right)\left(\frac{\boldsymbol{X}_{\boldsymbol{1}:\lambda}^{(\boldsymbol{g}+\boldsymbol{1})} - \boldsymbol{M}^{\boldsymbol{g}}}{\sigma^{(g)}}\right)^{\boldsymbol{T}} \qquad (4)$$

2.3. Step Size Control

The Covariance matrix adaption above does not control the overall scale of the distribution, i.e. Step-Size. The covariance matrix increases the scale in only one direction for each selected step, and can decrease the scale only by fading out old information via the factor 1- C1 - Cμ less information. To control the step-size $\sigma^{(g)}$ we exploit the evolution path and this can be applied independent of the covariance matrix adaption and is denoted as Cumulative step length adaption (CSA) as below.

- Whenever the evolution path is short, single steps cancel each other. In this case the step size should be decreased.

- Whenever the evolution path is long and single steps are pointing in the same direction, the step size should be increased to reduce the number of steps.

To decide whether the evolution path is long or short, the path length is compared with expected length under random selection. If the selection path biases the evolution path to be longer than expected, σ is increased, and if the selection path biases the evolution path to be shorter than expected, σ in decreased.

3. IMPLEMENTATION

3.1. Pseudo Code

Presented below is the pseudo code followed by detailed explanation of the major steps in the algorithm [6] [7].

```
Step 1: Initialize the CMA-ES Parameters
            D            ← no. of dimensions
            λ            ← Offspring population size (4.0 + 3.0 Log(D))
            μ            ← Parent population for next generation (floor(λ/2) )
            $\sigma_{Start}$       ← Initial standard deviation.
            $c_{cov}$         ← Covariance learning rate

Step 2: While stopping criterion is not met do
Step 3:         Update the Covariance Matrix $C^{(g+1)}$ (see Covariance Matrix Adaption)

    $$\boldsymbol{C^{(g+1)}} \leftarrow (\boldsymbol{1 - c_{cov}})\boldsymbol{C^g} + \frac{\boldsymbol{c_{cov}}}{\boldsymbol{\mu_{cov}}} \boldsymbol{P_c^{(g+1)} P_c^{(g+1)^T}} + \boldsymbol{c_{cov}}\left(\boldsymbol{1} - \frac{\boldsymbol{1}}{\boldsymbol{\mu_{cov}}}\right) \times \sum_{\boldsymbol{i=1}}^{\times} \boldsymbol{w_i} \left(\frac{\boldsymbol{X_{1:\lambda}^{(g+1)} - M^g}}{\sigma^{(g)}}\right)\left(\frac{\boldsymbol{X_{1:\lambda}^{(g+1)} - M^g}}{\sigma^{(g)}}\right)^{\boldsymbol{T}}$$

Step 4:         Update the Step Size $\sigma^g$ (see Step Size adaption)

    $$\sigma_{g+1} \leftarrow \sigma_g \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||P_\sigma||}{E\ N(0,I)} - 1\right)\right)$$

Step 5:         Generate Sample Population for generation g+1
                    $x_k^{(g+1)} \sim N\big(M^{(g)}, (\sigma^{(g)})^2\ C^{(g)}\big)$   for k = 1,…,λ
Step 6:         Update the mean for generation g+1
                    $m^{(g+1)} \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)}$
Step 7:         Update best ever solution
Step 8:     End While
```

Figure 2. CMA-ES Pseudo Code

The CMA-ES algorithm consists of 3 major steps as described above. For any algorithm, the stopping criteria should be an integral part and for CMA-ES it can be specified in the following ways:

1) Running the Algorithm for a pre-defined number of generations.
2) Stopping any further execution when the algorithm does not make any growth in approaching the solution.
3) Stopping when the Algorithm reaches a pre-defined objective function value.

3.1.1. Generate Sample Population

The population of a new search point at generation (g+1) is generated by sampling a multi-variate of the normal distribution with mean $M^{(g)}$ using Equation (1).

$$x_k^{(g+1)} \sim N\left(M^{(g)}, (\sigma^{(g)})^2\, C^{(g)}\right) \quad \text{for k = 1,…,λ}$$

3.1.2. Selection and Recombination

The new mean of the search distribution for Generation g+1 is the weighted average of best μ selected children from λ sample points generated by Equation (1).

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)} \tag{5}$$

$$\sum_{i=1}^{\mu} w_i = 1 \qquad w_i > 0 \quad for \quad w_i = 1, \dots, \mu \tag{6}$$

The individual weights for the weight vector are defined based on μ_{eff} such that $1 < \mu_{eff} < \mu$. In our case we choose $\mu_{eff} \approx \lambda/4$, which is a reasonable setting for w_i.

$$\mu_{eff} = \left(\sum_{i=1}^{\mu} {w_i}^2\right)^{-1} \tag{7}$$

3.1.3. Covariance Matrix Adaption

The goal of this step is to calculate the Covariance matrix at the generation g. Below are some of the parameters that are used in the process of calculating the Covariance matrix. The current generation Covariance matrix depends on the learning curve based on the previous Covariance matrix.

$$\boldsymbol{C^{(g+1)} = (1 - c_{cov})C^{g} + \frac{c_{cov}}{\mu_{cov}} P_c^{(g+1)} {P_c^{(g+1)}}^T + c_{cov}\left(1 - \frac{1}{\mu_{cov}}\right)}$$

$$\times \sum_{i=1}^{\times} \boldsymbol{w_i} \left(\frac{\boldsymbol{X_{1:\lambda}^{(g+1)} - M^{g}}}{\sigma^{(g)}}\right)\left(\frac{\boldsymbol{X_{1:\lambda}^{(g+1)} - M^{g}}}{\sigma^{(g)}}\right)^{\boldsymbol{T}} \qquad (8)$$

Where

$\boldsymbol{c_{cov}}$ is a strategy parameter equal to $min(\mu_{cov},\ \mu_{eff}, n^2) / n^2$

$\mu_{\boldsymbol{cov}}$ is > 0 and $\mu_{\boldsymbol{cov}} \sim \mu_{eff}$ in most cases. In our case $\mu_{eff} = 3.4$

$\boldsymbol{P_c^{(g+1)}}$ is the evolution path. As a sequence of steps the strategy takes over a number of generations. The evolution path can be derived as:

$$P_C^{(g+1)} = (1 - c_c)P_C^{(g)} + \sqrt{c_c(2 - c_c)\mu_{eff}}\ \frac{M^{(g+1)} - M^{g}}{\sigma^{(g)}} \qquad (9)$$

3.1.4. Variance Adaption

Along with the Covariance Matrix adaption, the Variance/Step Size is updated every generation. It is updated using the cumulative step size adaption (CSA) and can be defined as:

$$\sigma_{g+1} = \sigma_g \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||P_\sigma||}{E\ N(0,I)} - 1\right)\right) \qquad (10)$$

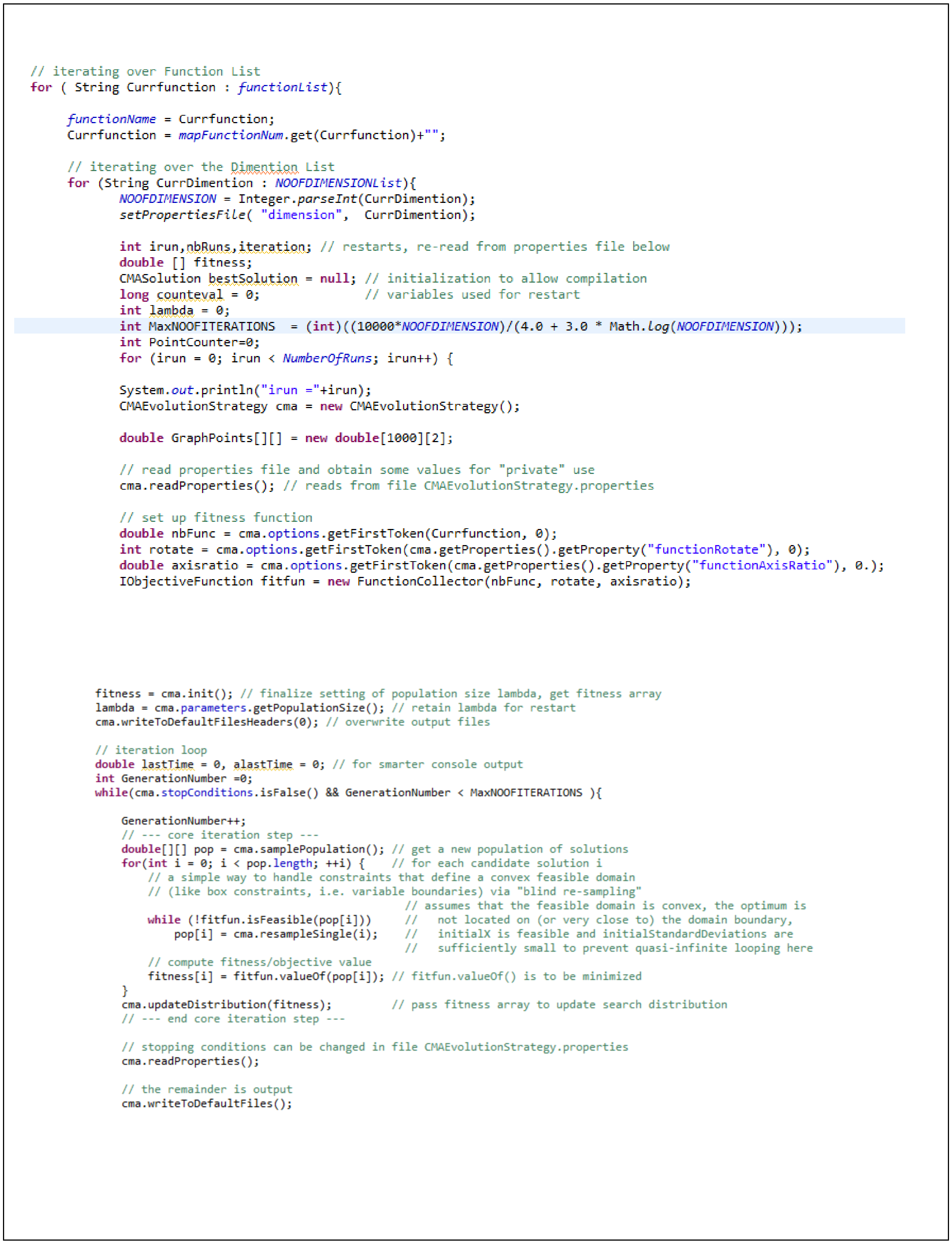
And P_σ is derived as:

$$P_\sigma = (1 - c_\sigma)P_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}\, C_K{}^{-1/2}\ \frac{M_{K+1} - M_{K+1}}{\sigma_k} \qquad (11)$$

$$\text{where } \mu_w = \left(\sum_{i=1}^{\mu} w_i{}^2\right)^{-1}$$

3.2. Source Code Overview

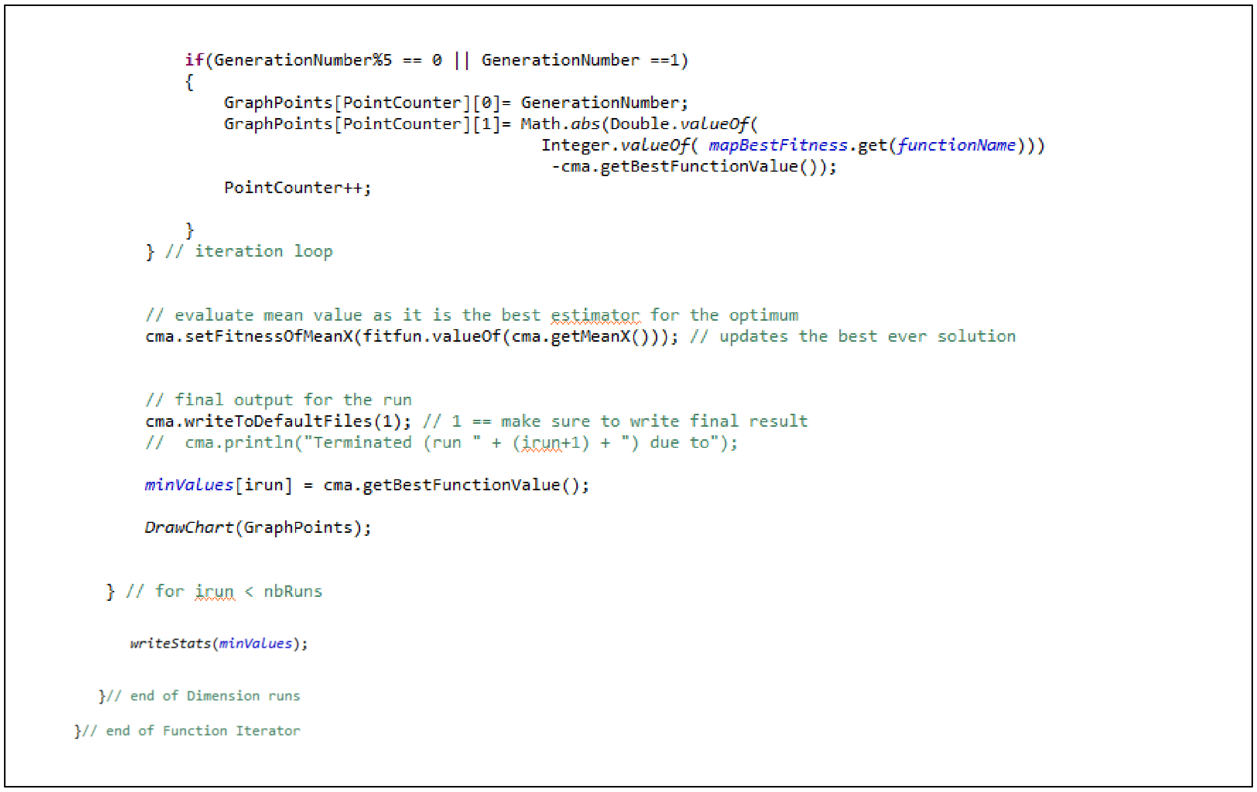
The CMA-ES Algorithm has been implemented in the JAVA Programming language. The package consists of 7 core classes and a properties file. The kernel is implemented in CMAES.java. The user can specify the number of dimensions, functions to be executed, number of runs, exit criteria and CMA-ES configuration parameters in the Properties file CMAEvolutionStrategy.properties. Any method that represents the benchmark function can be added to the class FunctionCollector.java. The kernel reads the arguments from the properties file, executes the Algorithm and reports the results. Below are the primary classes and their purpose.



```
// iterating over Function List
for ( String Currfunction : functionList){

    functionName = Currfunction;
    Currfunction = mapFunctionNum.get(Currfunction)+"";

    // iterating over the Dimention List
    for (String CurrDimention : NOOFDIMENSIONList){
        NOOFDIMENSION = Integer.parseInt(CurrDimention);
        setPropertiesFile( "dimension",  CurrDimention);

        int irun,nbRuns,iteration; // restarts, re-read from properties file below
        double [] fitness;
        CMASolution bestSolution = null; // initialization to allow compilation
        long counteval = 0;              // variables used for restart
        int lambda = 0;
        int MaxNOOFITERATIONS  = (int)((10000*NOOFDIMENSION)/(4.0 + 3.0 * Math.log(NOOFDIMENSION)));
        int PointCounter=0;
        for (irun = 0; irun < NumberOfRuns; irun++) {

        System.out.println("irun ="+irun);
        CMAEvolutionStrategy cma = new CMAEvolutionStrategy();

        double GraphPoints[][] = new double[1000][2];

        // read properties file and obtain some values for "private" use
        cma.readProperties(); // reads from file CMAEvolutionStrategy.properties

        // set up fitness function
        double nbFunc = cma.options.getFirstToken(Currfunction, 0);
        int rotate = cma.options.getFirstToken(cma.getProperties().getProperty("functionRotate"), 0);
        double axisratio = cma.options.getFirstToken(cma.getProperties().getProperty("functionAxisRatio"), 0.);
        IObjectiveFunction fitfun = new FunctionCollector(nbFunc, rotate, axisratio);
```

```
        fitness = cma.init(); // finalize setting of population size lambda, get fitness array
        lambda = cma.parameters.getPopulationSize(); // retain lambda for restart
        cma.writeToDefaultFilesHeaders(0); // overwrite output files

        // iteration loop
        double lastTime = 0, alastTime = 0; // for smarter console output
        int GenerationNumber =0;
        while(cma.stopConditions.isFalse() && GenerationNumber < MaxNOOFITERATIONS ){

            GenerationNumber++;
            // --- core iteration step ---
            double[][] pop = cma.samplePopulation(); // get a new population of solutions
            for(int i = 0; i < pop.length; ++i) {    // for each candidate solution i
                // a simple way to handle constraints that define a convex feasible domain
                // (like box constraints, i.e. variable boundaries) via "blind re-sampling"
                                                       // assumes that the feasible domain is convex, the optimum is
                while (!fitfun.isFeasible(pop[i]))     //   not located on (or very close to) the domain boundary,
                    pop[i] = cma.resampleSingle(i);    //   initialX is feasible and initialStandardDeviations are
                                                       //   sufficiently small to prevent quasi-infinite looping here
                // compute fitness/objective value
                fitness[i] = fitfun.valueOf(pop[i]); // fitfun.valueOf() is to be minimized
            }
            cma.updateDistribution(fitness);         // pass fitness array to update search distribution
            // --- end core iteration step ---

            // stopping conditions can be changed in file CMAEvolutionStrategy.properties
            cma.readProperties();

            // the remainder is output
            cma.writeToDefaultFiles();
```

Figure 3. Source Code Overview



```
                if(GenerationNumber%5 == 0 || GenerationNumber ==1)
                {
                    GraphPoints[PointCounter][0]= GenerationNumber;
                    GraphPoints[PointCounter][1]= Math.abs(Double.valueOf(
                                                    Integer.valueOf( mapBestFitness.get(functionName)))
                                                     -cma.getBestFunctionValue());
                    PointCounter++;

                }
            } // iteration loop


            // evaluate mean value as it is the best estimator for the optimum
            cma.setFitnessOfMeanX(fitfun.valueOf(cma.getMeanX())); // updates the best ever solution


            // final output for the run
            cma.writeToDefaultFiles(1); // 1 == make sure to write final result
            //  cma.println("Terminated (run " + (irun+1) + ") due to");

            minValues[irun] = cma.getBestFunctionValue();

            DrawChart(GraphPoints);


        } // for irun < nbRuns

          writeStats(minValues);

      }// end of Dimension runs

    }// end of Function Iterator
```

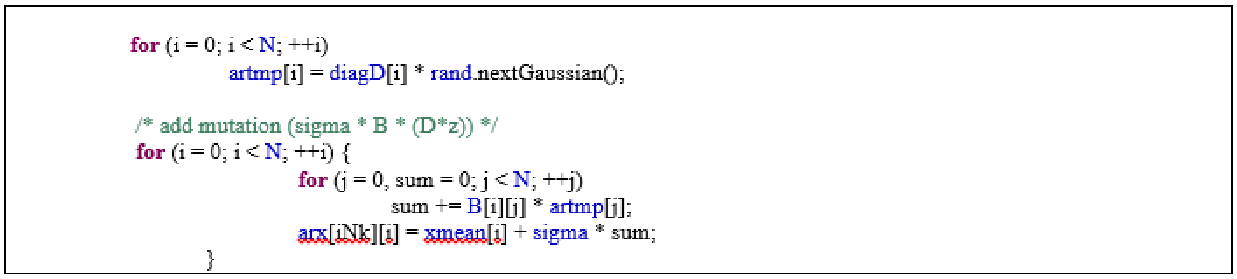
Figure 3. Source Code Overview (continued)

3.2.1. CMAES.java

This class contains the main method that reads the properties file for properties such as Number of Runs, Benchmark Function list, Dimensions to be executed, and then calls the CMAEvolutionStrategies class, Writing the stats file and drawing the graphs.

3.2.2. CMAEvolutionStrategy.java

This is the core of the application with below methods.



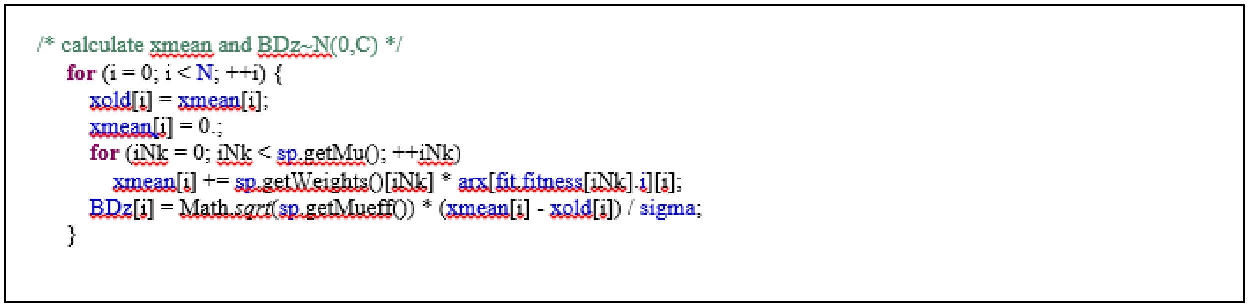
```
for (i = 0; i < N; ++i)
        artmp[i] = diagD[i] * rand.nextGaussian();

/* add mutation (sigma * B * (D*z)) */
 for (i = 0; i < N; ++i) {
                for (j = 0, sum = 0; j < N; ++j)
                        sum += B[i][j] * artmp[j];
                arx[iNk][i] = xmean[i] + sigma * sum;
        }
```

Figure 4. CMA-ES Core logic

SamplePopulation(): This method generates sample population from Mean Vector at generation G, Standard Deviation at generation G and Covariance matrix at Generation G.

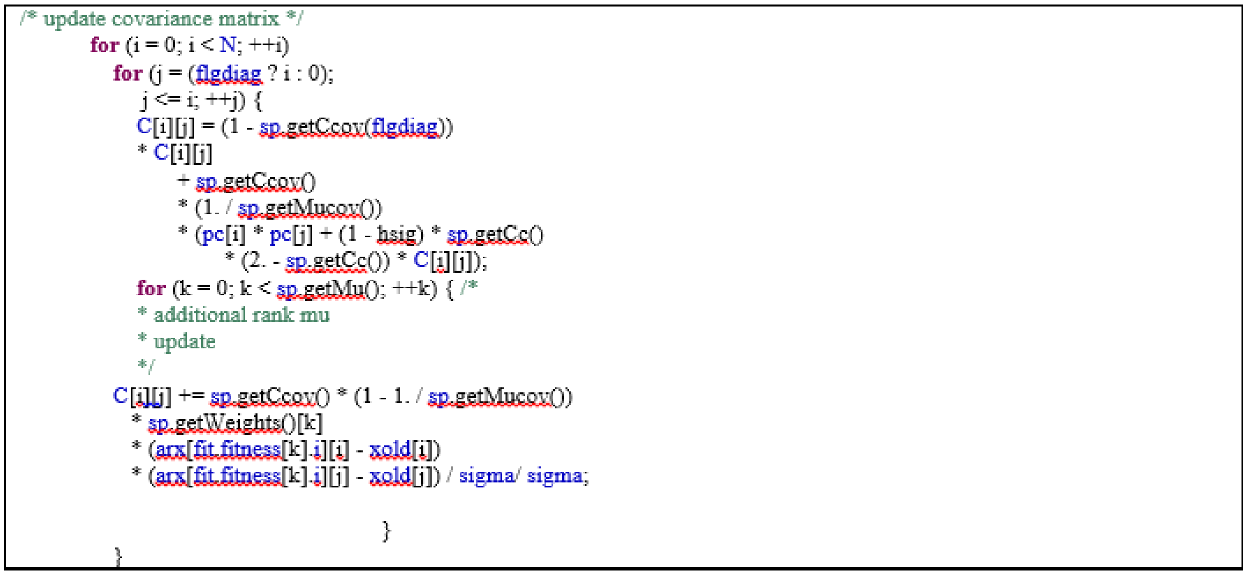
3.2.3. UpdateDistribution

This method generates the Mean Vector Xmean to be used in generation G+1 and updates the covariance matrix C for generation G+1.



```
/* calculate xmean and BDz~N(0,C) */
    for (i = 0; i < N; ++i) {
      xold[i] = xmean[i];
      xmean[i] = 0.;
      for (iNk = 0; iNk < sp.getMu(); ++iNk)
        xmean[i] += sp.getWeights()[iNk] * arx[fit.fitness[iNk].i][i];
      BDz[i] = Math.sqrt(sp.getMueff()) * (xmean[i] - xold[i]) / sigma;
    }
```

Figure 5. Mean Update



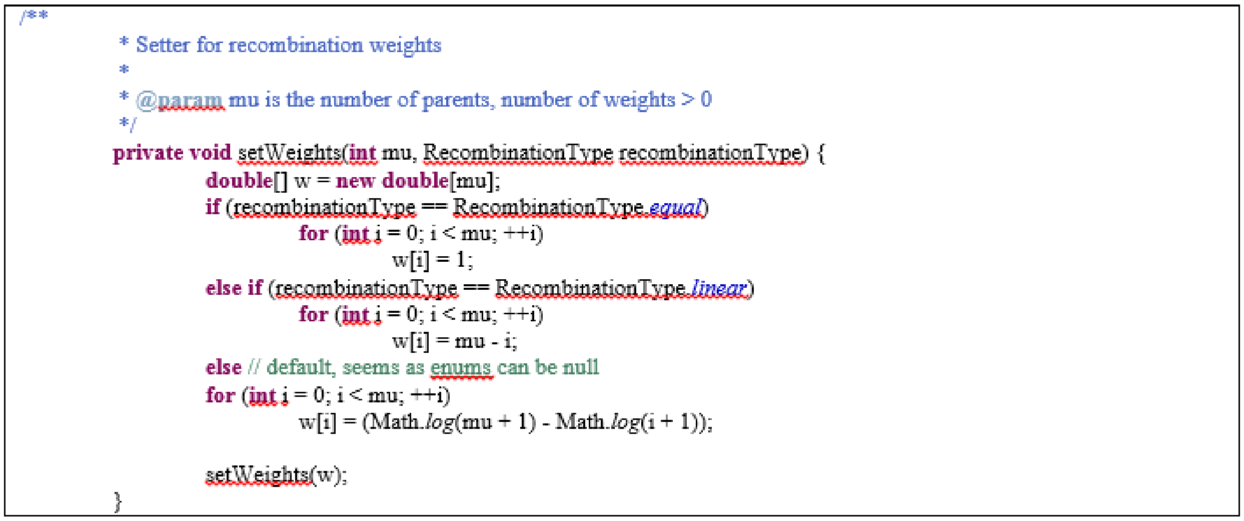
```
/* update covariance matrix */
        for (i = 0; i < N; ++i)
          for (j = (flgdiag ? i : 0);
              j <= i; ++j) {
              C[i][j] = (1 - sp.getCcov(flgdiag))
              * C[i][j]
                  + sp.getCcov()
                  * (1. / sp.getMucov())
                  * (pc[i] * pc[j] + (1 - hsig) * sp.getCc()
                      * (2. - sp.getCc()) * C[i][j]);
              for (k = 0; k < sp.getMu(); ++k) { /*
              * additional rank mu
              * update
              */
          C[i][j] += sp.getCcov() * (1 - 1. / sp.getMucov())
            * sp.getWeights()[k]
            * (arx[fit.fitness[k].i][i] - xold[i])
            * (arx[fit.fitness[k].i][j] - xold[j]) / sigma/ sigma;

                                  }
          }
```

Figure 6. Coveriance Matrix Update

3.2.4. CMAParameters.java

This class is used to store and retrieve all the CMAES parameters read from the CMAEvolutionStrategies.properties file throughout the execution. Major methods include:

setWeights(): setting the weight vector based on the recombination type.



```
/**
         * Setter for recombination weights
         *
         * @param mu is the number of parents, number of weights > 0
         */
        private void setWeights(int mu, RecombinationType recombinationType) {
                double[] w = new double[mu];
                if (recombinationType == RecombinationType.equal)
                        for (int i = 0; i < mu; ++i)
                                w[i] = 1;
                else if (recombinationType == RecombinationType.linear)
                        for (int i = 0; i < mu; ++i)
                                w[i] = mu - i;
                else // default, seems as enums can be null
                for (int i = 0; i < mu; ++i)
                        w[i] = (Math.log(mu + 1) - Math.log(i + 1));

                setWeights(w);
        }
```

Figure 7. Weight Vector

4. EXPERIMENTAL SETUP

4.1. IEEE CEC' 2013 Test Suite

Optimization problems are real world problems that we come across in the field of Mathematics, Science, Engineering, Business and Economics. Various factors contribute to the problems faced in searching the optimal solution. Firstly, with the increase in the number of variables associated with a problem the search space grows exponentially. Secondly, the properties of the problem tend to change as the dimensionality of the problem increases. Thirdly, computation of such large-scale problems is expensive. The IEEE CEC' 2013 [9] Test Suite 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 overview, the functions are described briefly. These functions are described in detail in [9]. The following terminology that is frequently used in the test suite is as follows:

$\boldsymbol{D}$ is the dimensionality of the problem.

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

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

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

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

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

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

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

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

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

Given these definitions, the functions are briefly described in the next section.

Table 1. 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 Schaffer’s F6 Function	600

4.1.1. Sphere Function

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

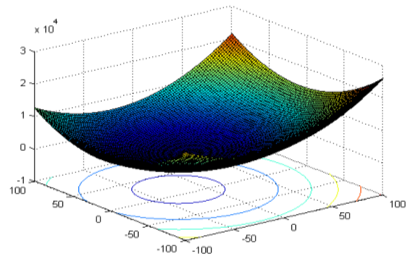


Figure 8. F1

Properties:

1 Unimodal

2 Separable

4.1.2. Rotated High Conditioned Elliptic Function

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

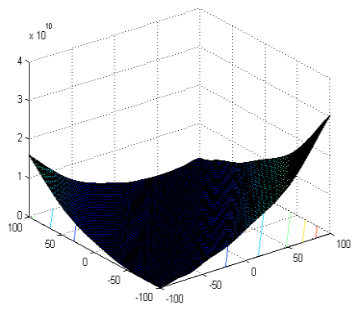


Figure 9. F2

Properties:

1 Unimodal

2 Non-Separable

4.1.3. Rotated Bent Cigar Function

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

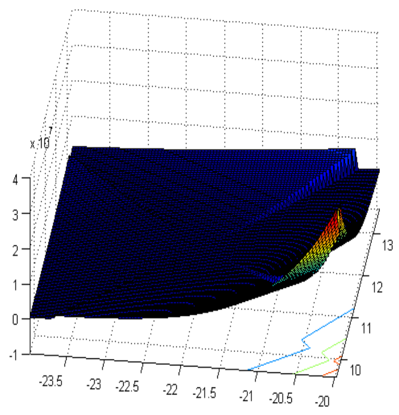


Figure 10. F4

Properties:

1 Unimodal

2 Non-Separable

3 Smooth but narrow ridge

4.1.4. Rotated Discus Function

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

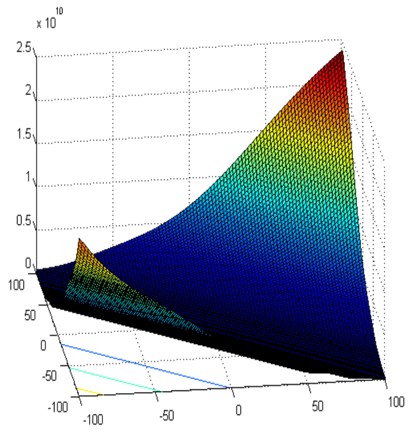


Figure 11. F4

Properties:

1 Unimodal

2 Separable

3 Asymmetrical

4 Smooth Local Irregularities

4.1.5. Different Powers Function

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

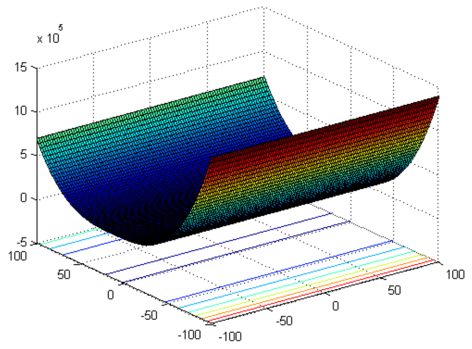


Figure 12. F5

Properties:

1 Unimodal

2 Separable

4.1.6. Rotated Rosenbrock's Function

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

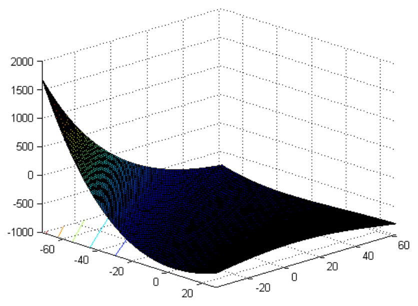


Figure 13. F6

Properties:

1 Multimodal

2 Non-Separable

3 Having a narrow valley from local optimum to global optimum

4.1.7. Rotated Schaffers F7 Function

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

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

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

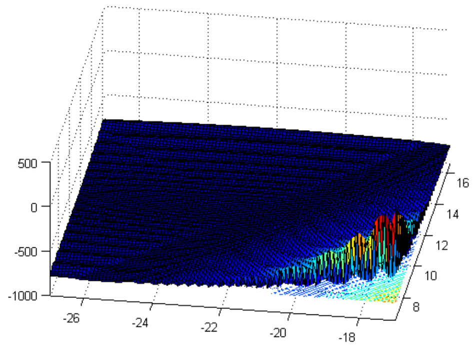


Figure 14. F7

Properties:

1 Multimodal

2 Non-Separable

3 Asymmetrical

4.1.8. Rotated Ackley's Function

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

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

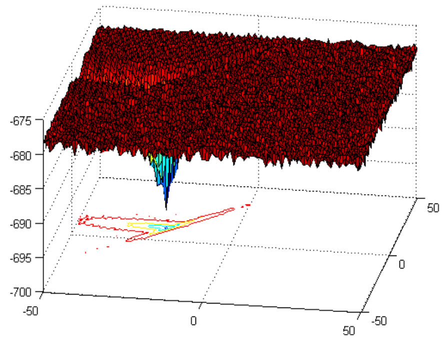


Figure 15. F8

Properties:

1 Multimodal

2 Non-separable

4.1.9. Asymmetrical Rotated Weierstrass Function

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

$$+f_9^*$$

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

$$and\ z = \Lambda^{10}M_2T_{asy}^{0.5}(M_1\frac{0.5}{100}(x-o))$$

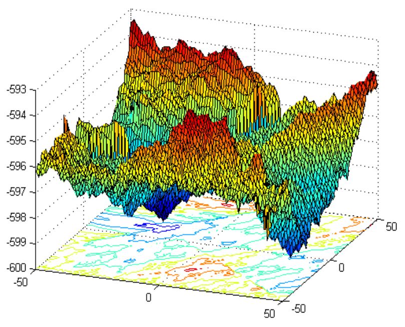


Figure 16. F9

Properties:

1 Multimodal

2 Non-Separable

3 Asymmetrical

4.1.10. Rotated Griewank's Function

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

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

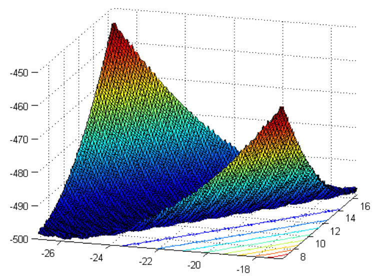


Figure 17. F10

Properties:

1 Multimodal

2 Non-separable

4.1.11. Rastrigin's Function

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

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

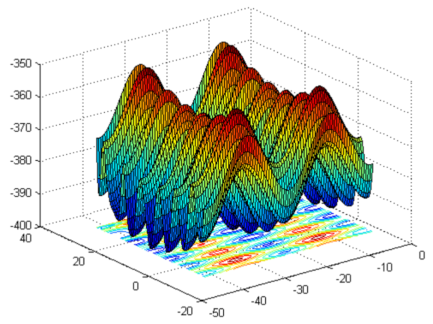


Figure 18. F11

Properties:

1 Multimodal

2 Non-separable

3 Huge number of local optima

4.1.12. Rotated Rastrigin's Function

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

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

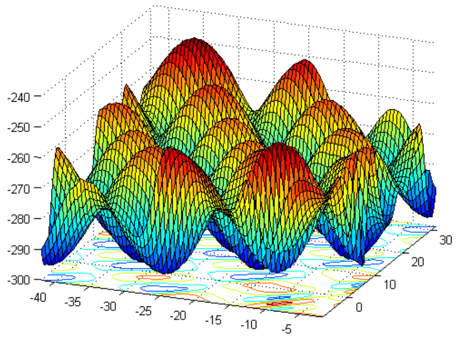


Figure 19. F12

Properties:

1 Multimodal

2 Non-separable

3 Asymmetrical

4 Huge number of local optima

4.1.13. Non-Continuous Rotated Rastrigin's Function

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

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

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

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

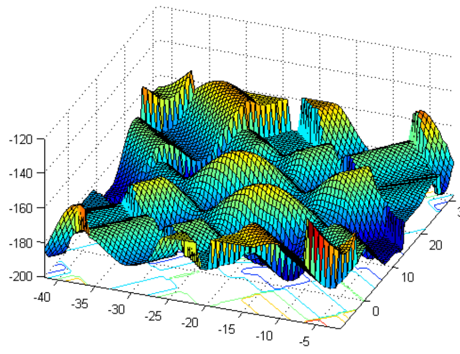


Figure 20. F13

Properties:

1 Multimodal

2 Asymmetrical

3 Huge number of local optima

4 Non-continuous

4.1.14. Schwefel's Function

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

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

$$g(z_i)$$

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

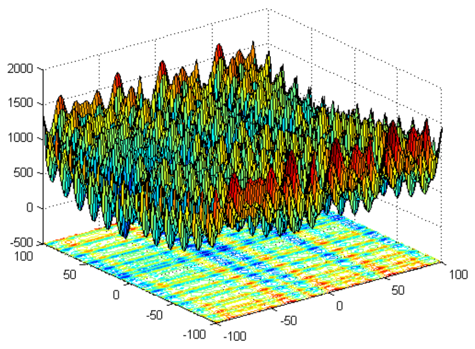


Figure 21. F14

Properties:

1 Multimodal

2 Non-separable

3 Huge number of local optima

4.1.15. Rotated Schwefel's Function

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

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

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

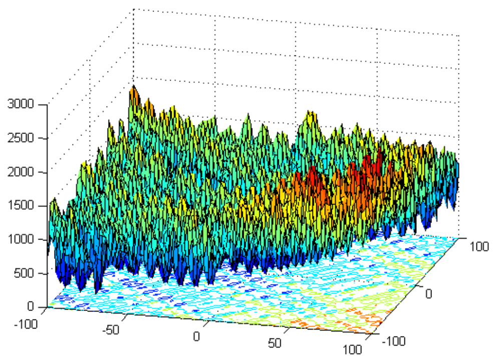


Figure 22. F15

Properties:

1 Multimodal

2 Asymmetrical

3 Non-separable

4.1.16. Rotated Katsuura Function

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

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

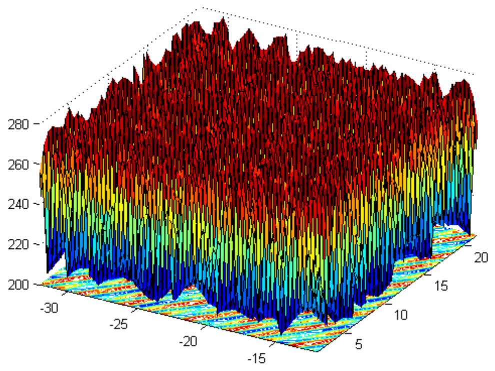


Figure 23. F16

Properties:

1 Multimodal

2 Non-separable

3 Asymmetrical

4.1.17. Lunacek Bi_Rastrigin Function

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

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

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

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

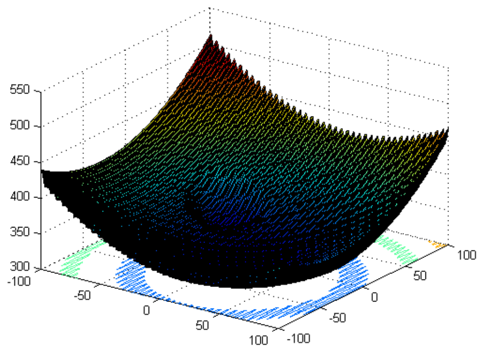


Figure 4. F17

Properties:

1 Multimodal

2 Asymmetrical

4.1.18. Rotated Lunacek Bi_Rastrigin Function

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

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

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

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

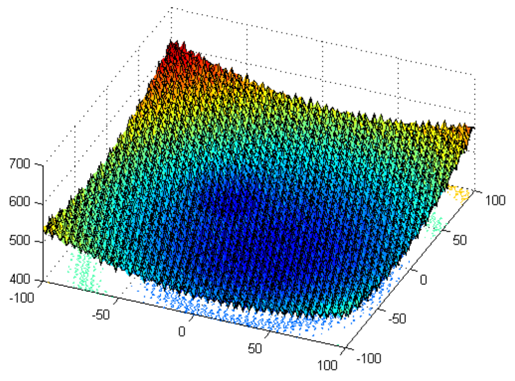


Figure 25. F18

Properties:

1 Multimodal

2 Non-separable

3 Asymmetrical

4.1.19. Expanded Griewank's plus Rosenbrock's Function

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

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

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

$$g_1\big(g_2(z_D, z_1)\big) + \ f_{19}^*$$

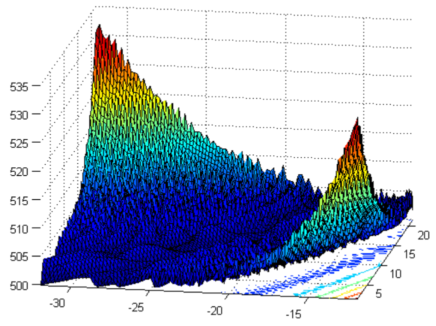


Figure 26. F19

Properties:

1 Multimodal

2 Asymmetrical

4.1.20. Expanded Scaffer's F6 Function

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

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

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

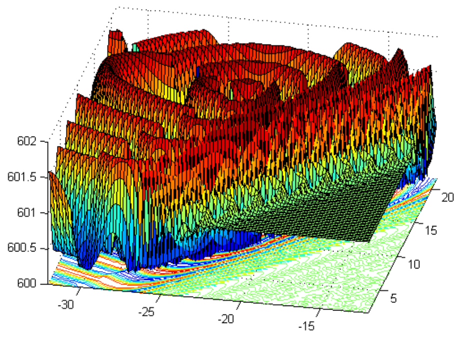


Figure 27. F20

Properties:

1 Multimodal

2 Non-separable

3 Asymmetrical

4.2. Experimental Settings

This paper evaluates the performance of the CMA-ES algorithm on the first 20 benchmark functions specified by the IEEE CEC' 2013 Test Suite. The algorithm is run on 3 different dimensions for each function namely 10D, 30D and 50D. Every function evaluation is run 50 times and the best, worst, mean, standard deviation values are recorded for each evaluation.

Since it is 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 criterion for stopping the algorithm: 1) number of evaluations reaches 10^4 times the problem dimensionality. This stopping criterion is significant as real world optimization is computationally intensive, thus, the predefined number of function evaluations serve a cutoff parameter to usually constrained computational 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}. Thus all the results with difference below 10^{-8} are represented as 0.0e+00.

5. RESULTS

Table 1 reports the performance of CMA-ES on benchmark functions at dimensionality 10D. The best, worst and mean values shown are achieved over 50 runs. For 10D problems, the algorithm is able to achieve the global minimum for 7 functions out of total of 20 functions.

Table 2. Performance of CMA-ES 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	0.00e+00	0.00e+00	0.00e+00
	Mean	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Std	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	0.00e+00	2.5e-02	6.25e-01	**4.67e-01**	0.00e+00
	Worst	3.98e+00	6.25e+02	2.15e+1	1.92e+01	2.8e-02
	Mean	1.01e+00	1.20e+03	2.03e+01	1.01e+01	0.01e+00
	Std	1.73e+00	2.22E+03	7.86e-02	5.72E+00	1.00e-02
		F11	**F12**	**F13**	**F14**	**F15**
	Best	**1.87e+02**	**2.90e+02**	**1.94e+01**	**1.11e+02**	**1.74e+03**
	Worst	2.31e+02	2.93e+02	5.07e+02	1.70e+02	2.27e+03
	Mean	2.22e+02	2.90e+02	2.83e+02	1.57e+03	2.05e+03
	Std	2.96e+01	5.70e-01	2.57e+02	1.88e+02	1.10e+02
		F16	F17	F18	F19	**F20**
	Best	**5.80e-02**	2.20e+02	2.40e+02	3.59E-01	**3.27e+00**
	Worst	1.42e+00	5.45e+02	3.80e+02	1.05e+00	5.00e+00
	Mean	2.49e-01	2.84e+02	2.62e+02	8.50e-01	4.63e+00
	Std	2.53e-01	9.80e+01	1.4e+02	2.98e-01	4.75e-01

Presented in Figure 22 through Figure 41 are the fitness versus generation plots of the CMA-ES algorithm at problem dimensionality of 10D. The initial population is set to 7, initial standard deviation is set to 0.5 and μ_{eff} = 3.4.

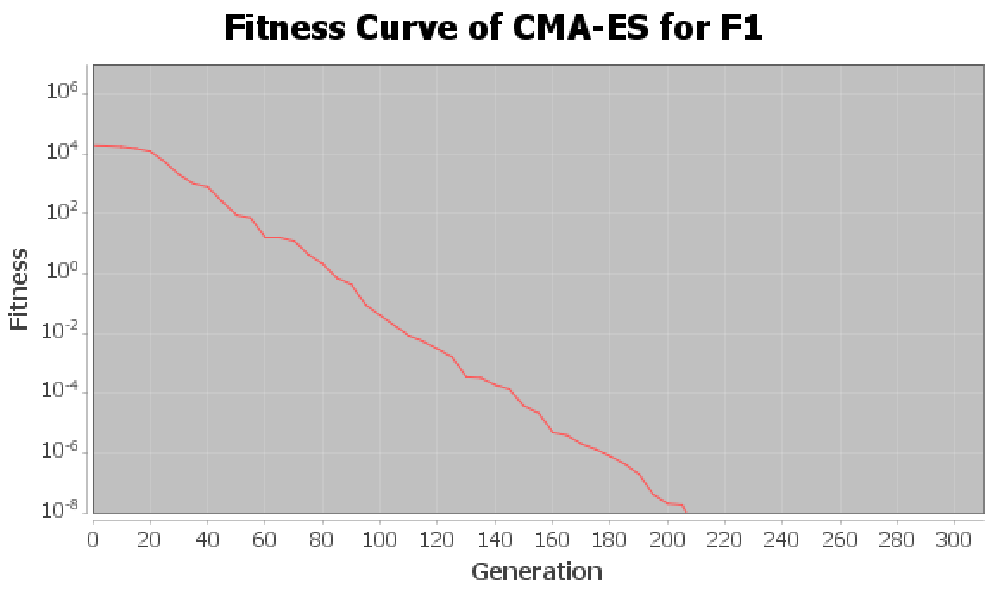


Figure 28. Average fitness curve of CMA-ES for F1

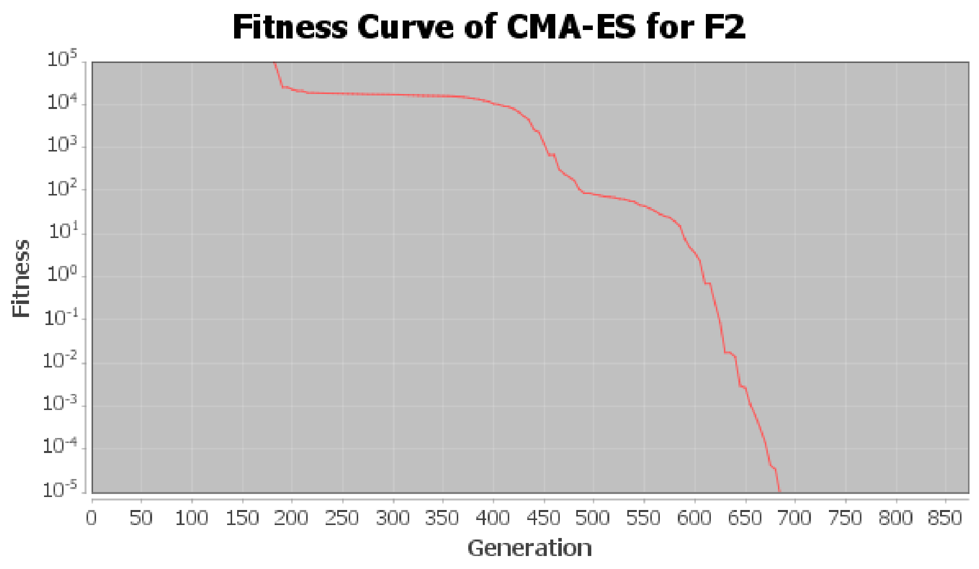


Figure 29. Average fitness curve of CMA-ES for F2

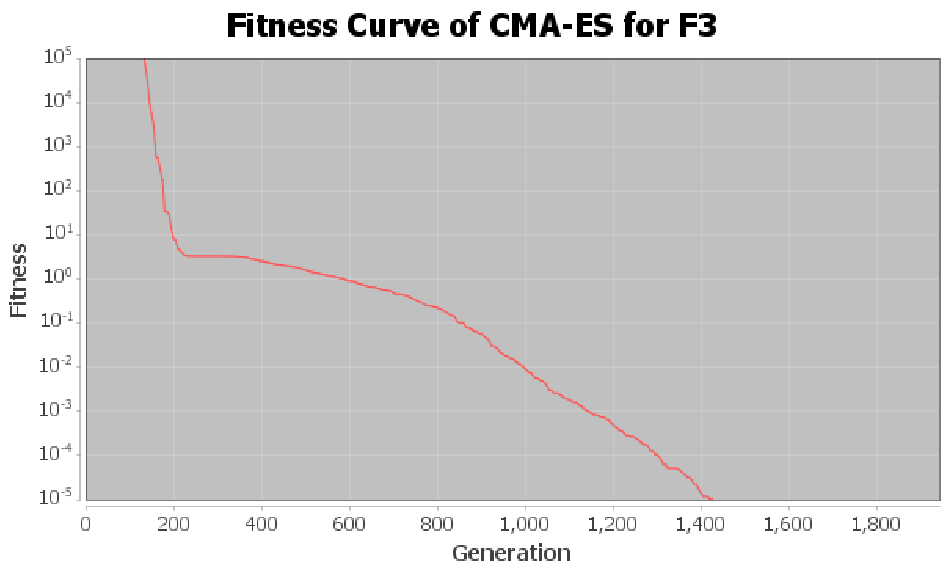


Figure 30. Average fitness curve of CMA-ES for F3

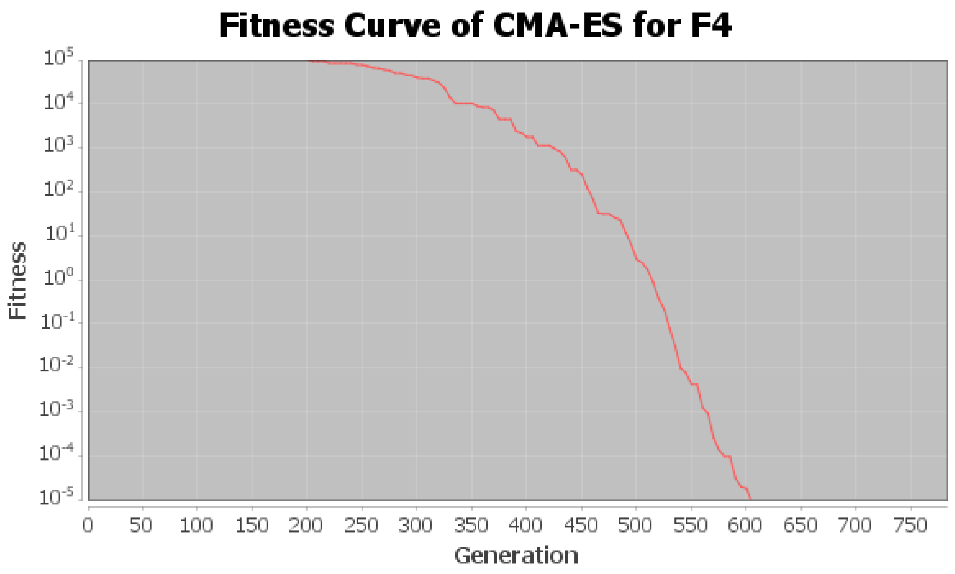


Figure 31. Average fitness curve of CMA-ES for F4

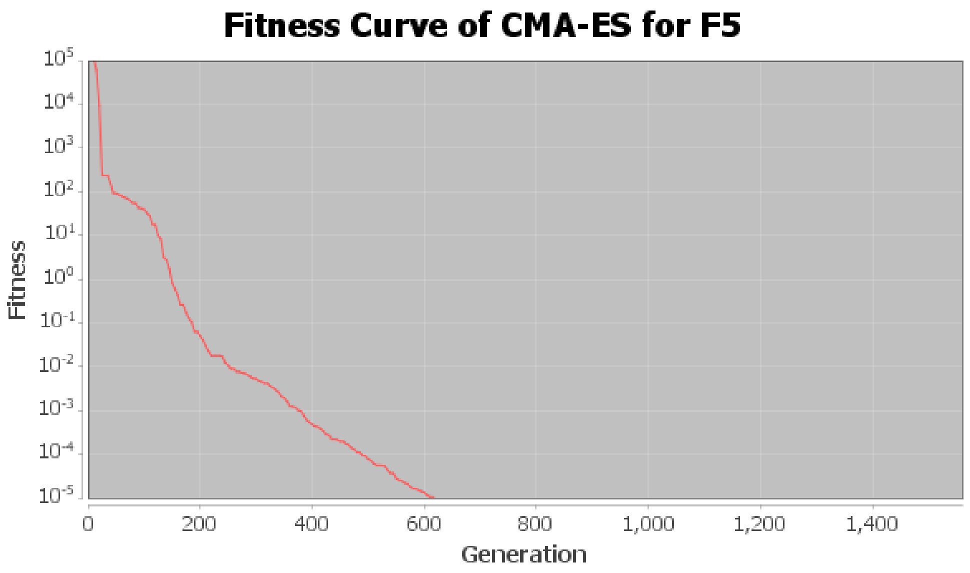


Figure 32. Average fitness curve of CMA-ES for F5

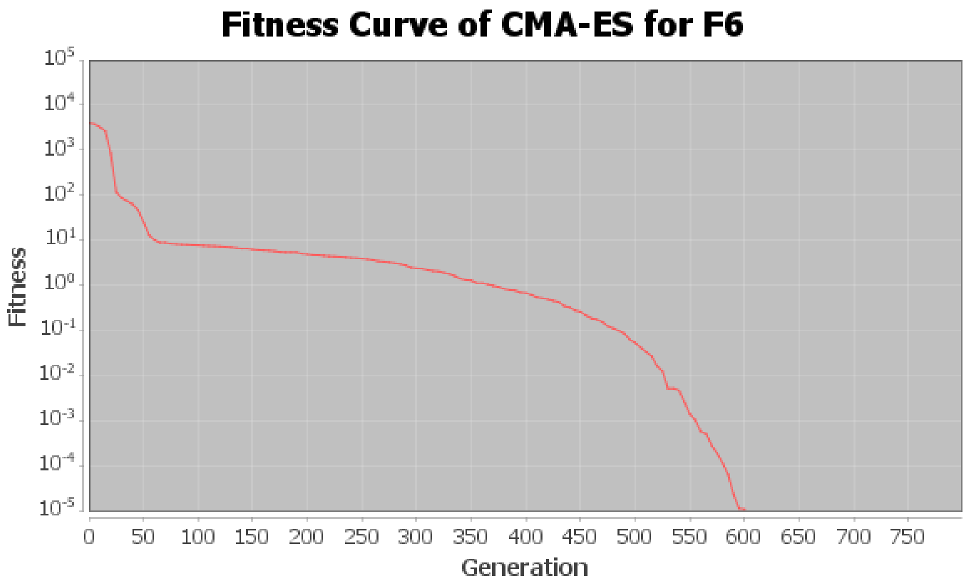


Figure 33. Average fitness curve of CMA-ES for F6

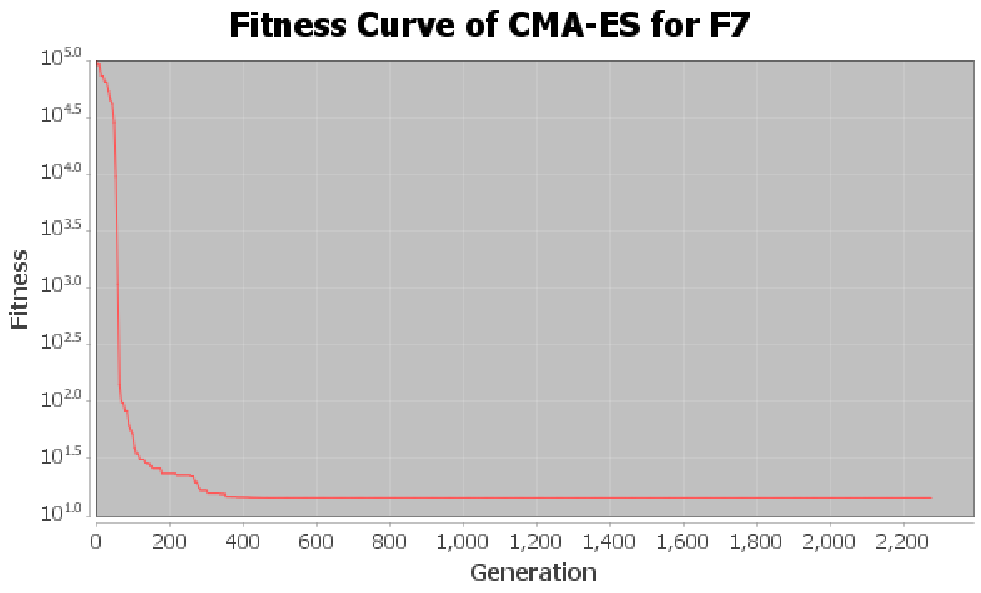


Figure 34. Average fitness curve of CMA-ES for F7

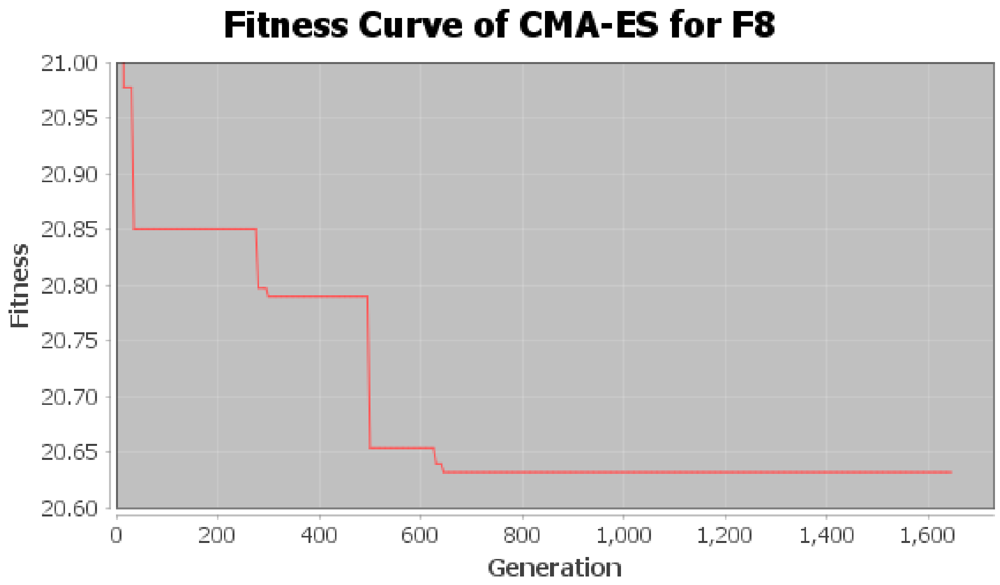


Figure 35. Average fitness curve of CMA-ES for F8

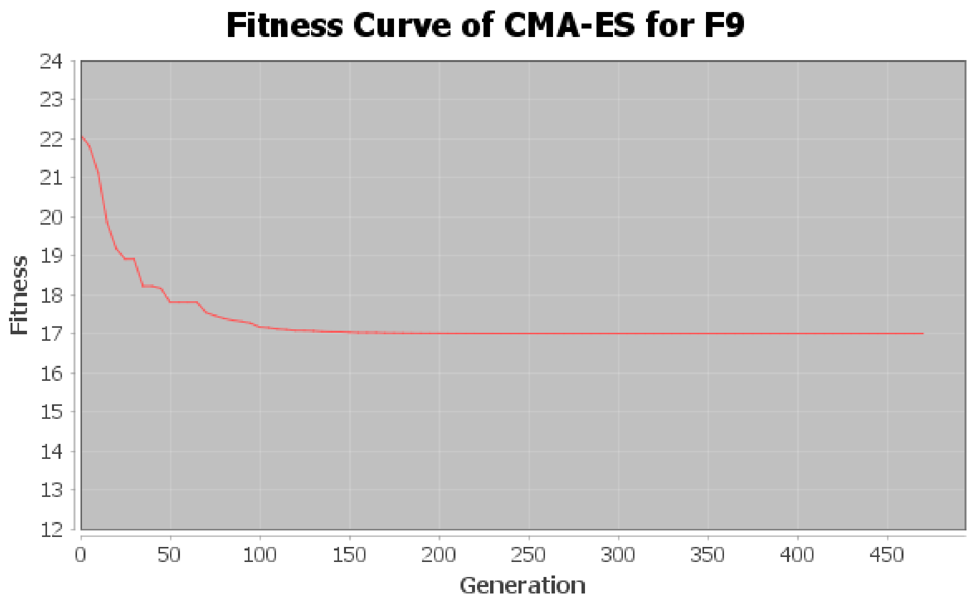


Figure 36. Average fitness curve of CMA-ES for F9

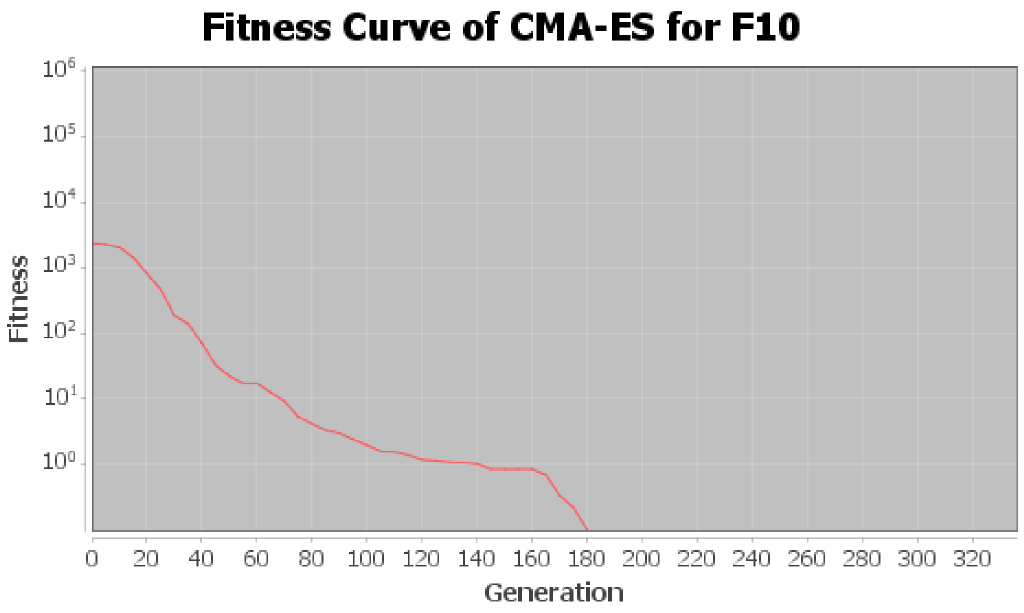


Figure 37. Average fitness curve of CMA-ES for F10

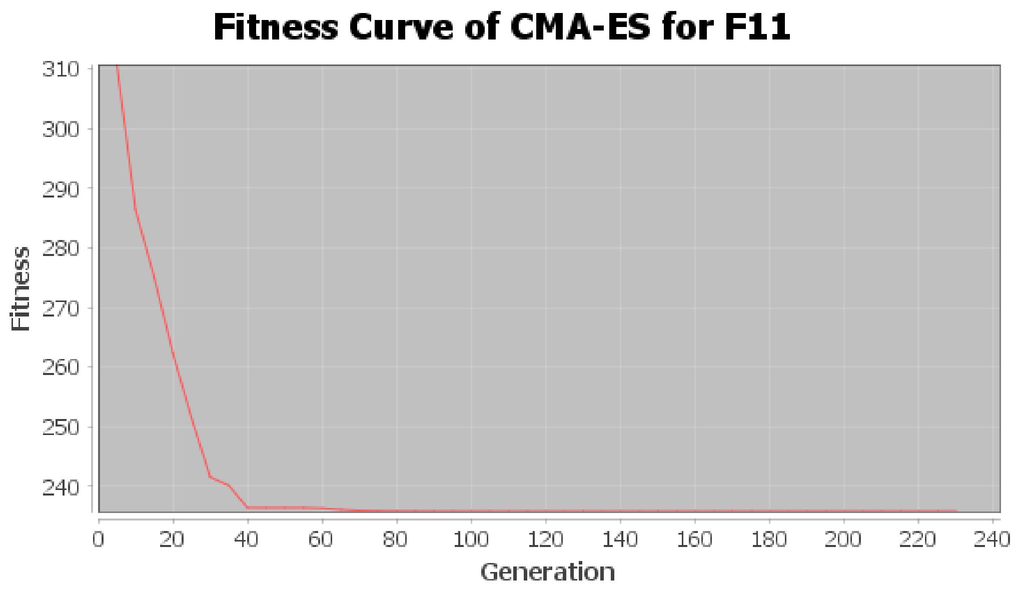


Figure 38. Average fitness curve of CMA-ES for F11

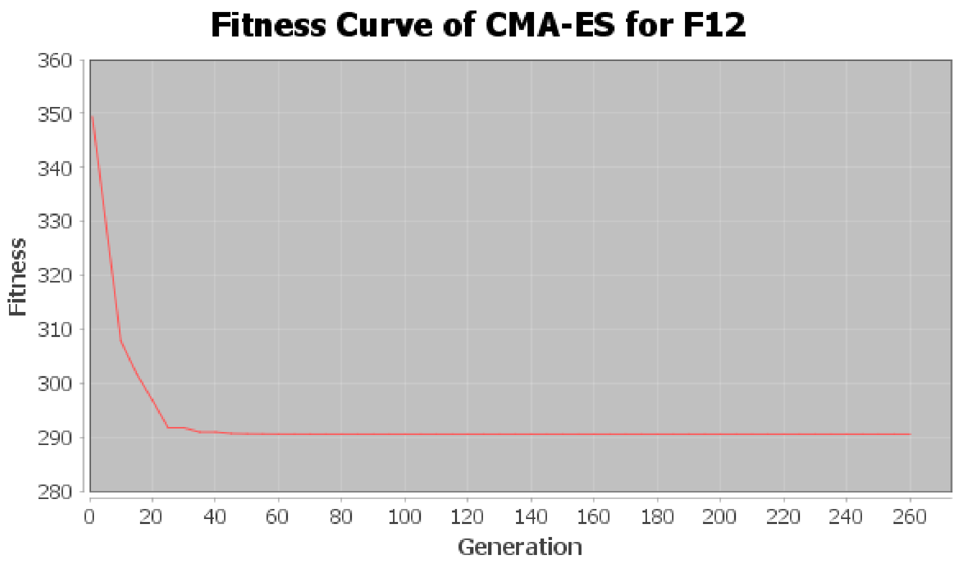


Figure 39. Average fitness curve of CMA-ES for F12

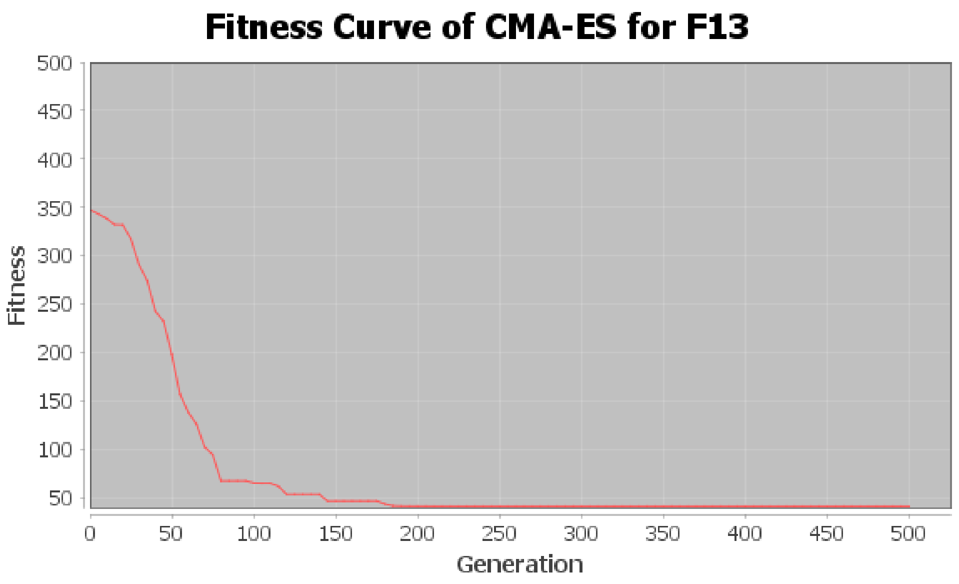


Figure 40. Average fitness curve of CMA-ES for F13

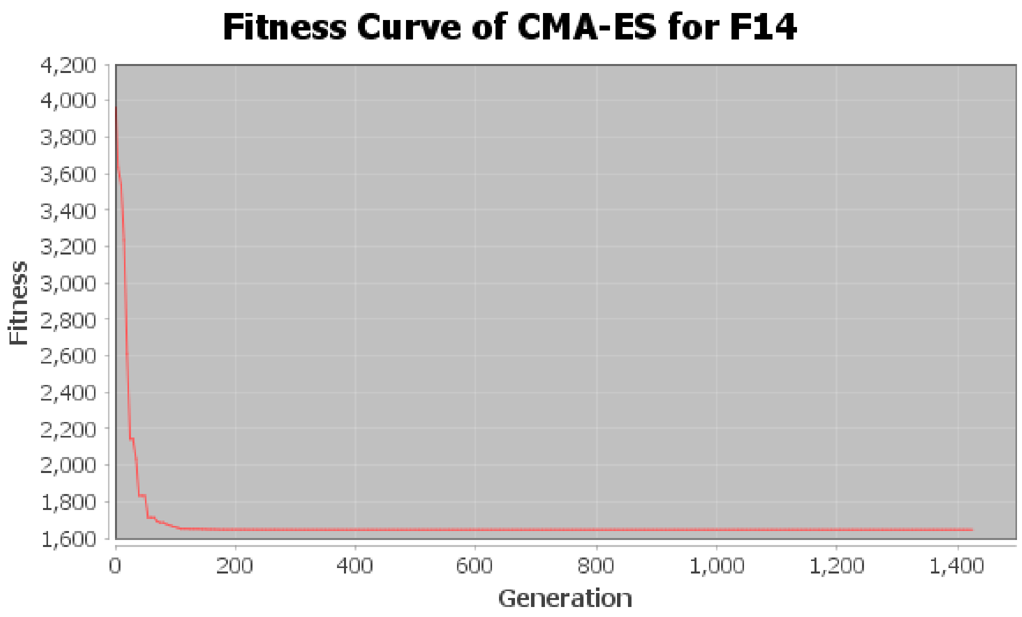


Figure 41. Average fitness curve of CMA-ES for F14

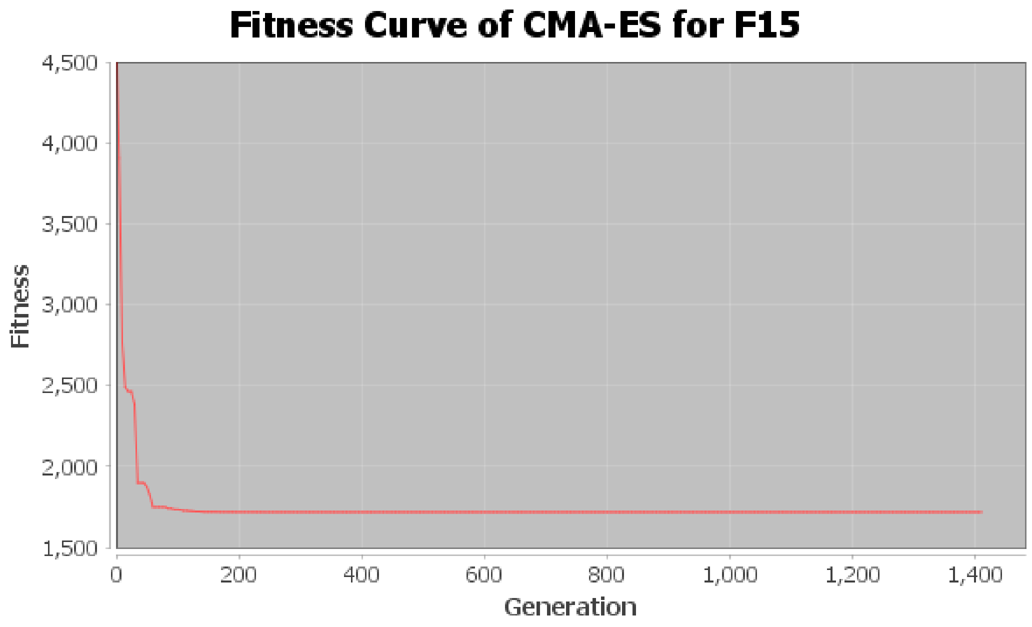


Figure 42. Average fitness curve of CMA-ES for F15

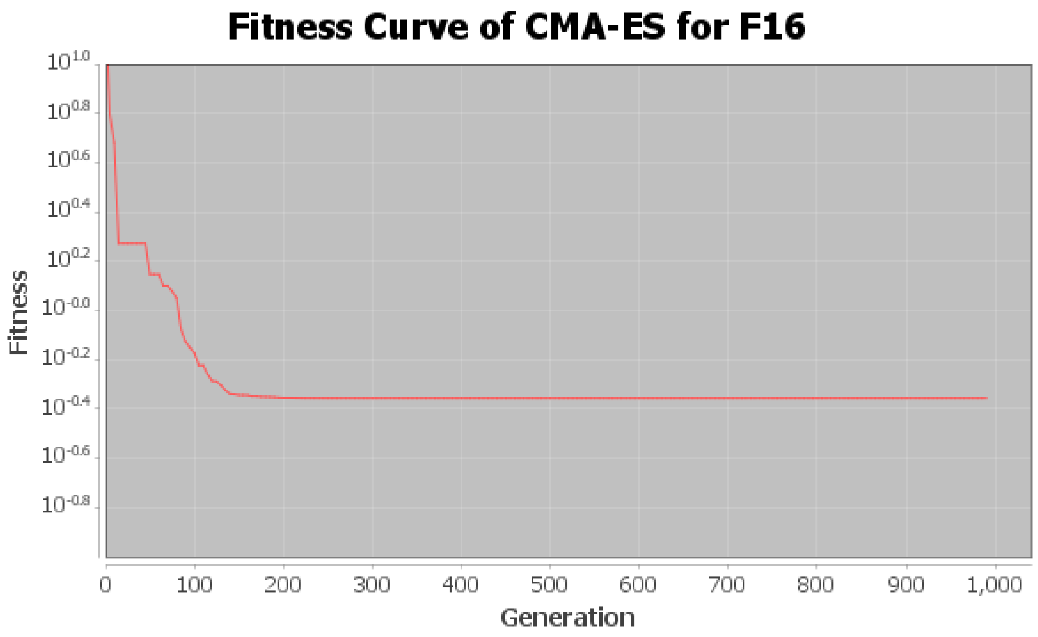


Figure 43. Average fitness curve of CMA-ES for F16

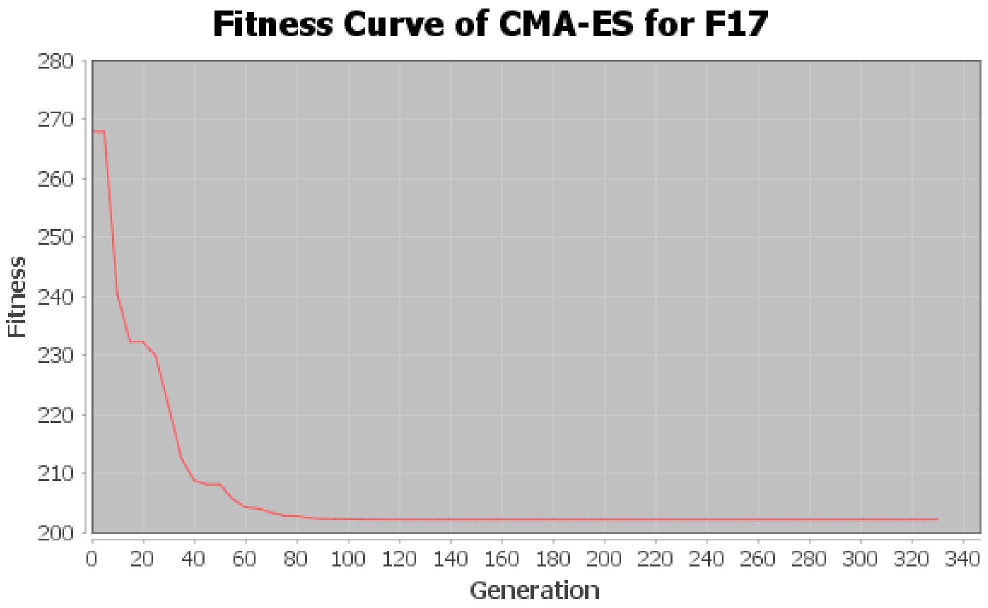


Figure 44. Average fitness curve of CMA-ES for F17

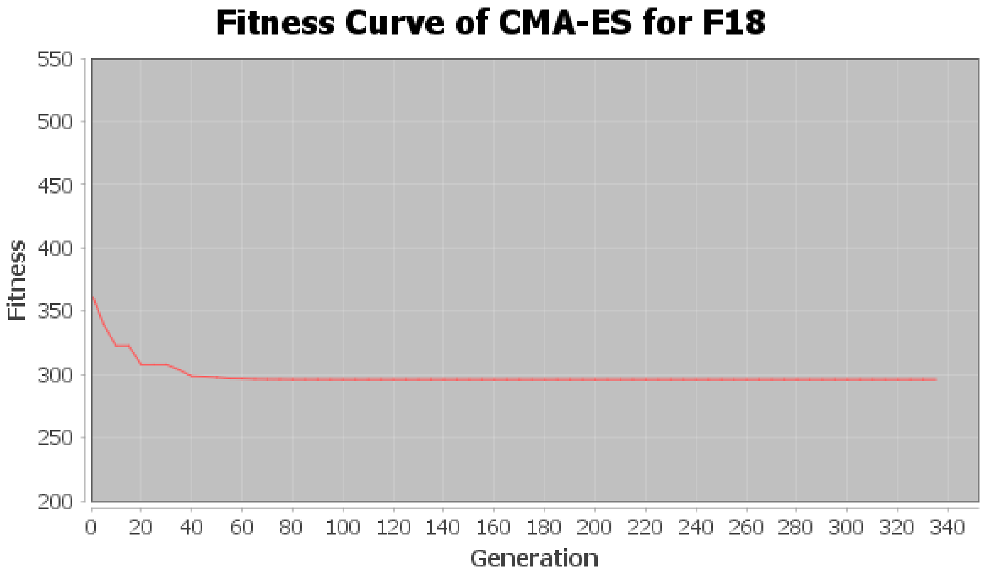


Figure 45. Average fitness curve of CMA-ES for F18

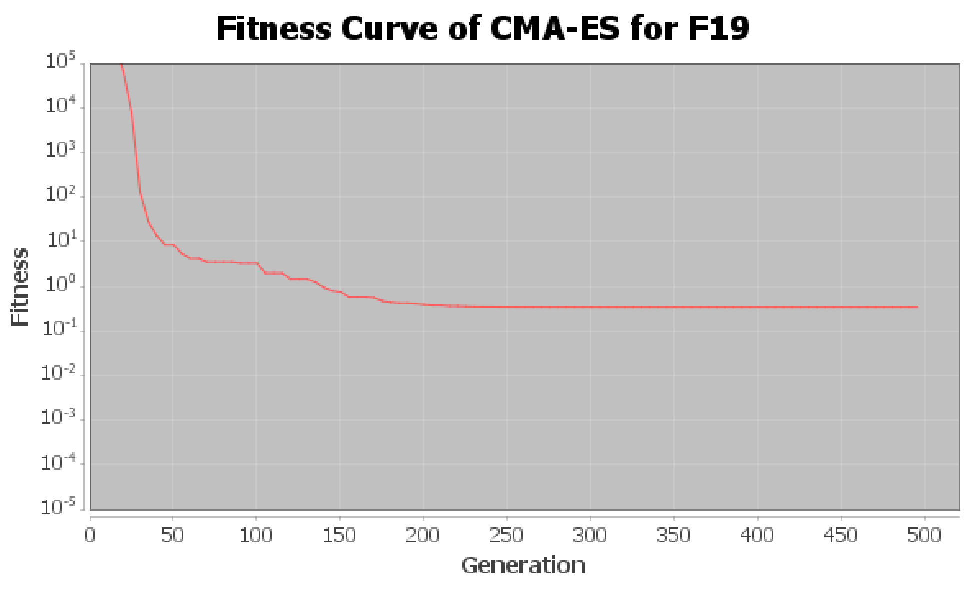


Figure 46. Average fitness curve of CMA-ES for F19

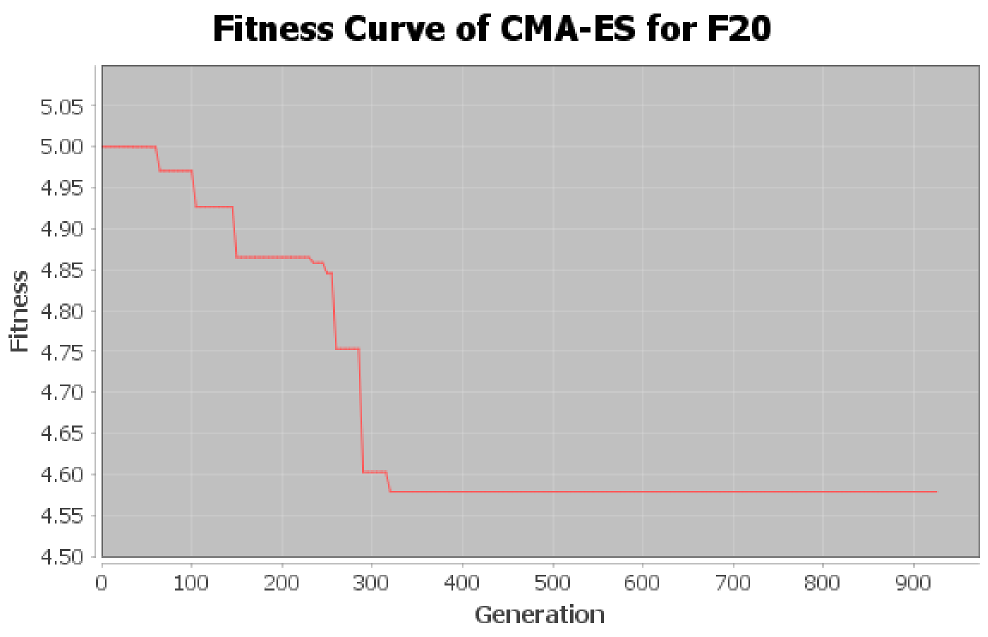


Figure 47. Average fitness curve of CMA-ES for F20

Table 3. Performance of *CMA-ES* at problem dimensionality 30

D	**PM**	**F1**	**F2**	**F3**	**F4**	**F5**
30 D	**Best**	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Worst	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Mean	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Std	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	0.00e+00	1.26e+00	2.09e+01	1.63e+01	0.00e+00
	Worst	3.98e+00	7.03e+01	7.00e+02	4.83e+01	9.11e-02
	Mean	2.34e-01	1.78e+01	2.03e+02	3.49e+01	1.49e-02
	Std	9.30e-01	1.34e+01	3.11e+02	7.52e+00	1.41e-02
		F11	**F12**	**F13**	**F14**	**F15**
	Best	3.28e+01	5.57e+01	8.30e+01	4.29e+03	3.95e+03
	Worst	9.50e+02	9.63e+02	2.42e+03	5.43e+03	5.33e+03
	Mean	4.63e+02	8.01e+02	1.21e+03	1.57e+03	4.67e+03
	Std	2.72e+02	2.69e+02	5.67e+02	1.88e+02	2.94e+02
		F16	**F17**	**F18**	**F19**	**F20**
	Best	2.21e-02	8.53e+02	1.39e+03	1.04e+00	1.27e+01
	Worst	1.34e-01	2.18e+03	1.41e+03	3.12e+00	1.50e+01
	Mean	6.26e-02	1.27e+03	1.40e+03	1.81e+00	1.49e+01
	Std	2.90e-02	2.49e+02	1.02e+02	4.51e-01	3.50e-01

Table 4. Performance of *CMA-ES* at problem dimensionality 50

D	**PM**	**F1**	**F2**	**F3**	**F4**	**F5**
50 D	**Best**	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Worst	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Mean	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	Std	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
		F6	**F7**	**F8**	**F9**	**F10**
	Best	0.00e+00	3.52e+00	2.10e+01	4.87e+01	0.00e+00
	Worst	3.98e+00	4.49e+01	7.00e+02	7.91e+01	5.42e-02
	Mean	4.01e-01	2.07e+01	2.03e+02	6.94e+01	1.95e-02
	Std	1.18e+01	8.79e+00	3.11e+02	8.97e+00	1.30e-02
		F11	**F12**	**F13**	**F14**	**F15**
	Best	7.31e+02	9.24e+02	1.42e+03	8.38e+03	6.46e+03
	Worst	9.69e+02	9.87e+02	5.87e+03	9.39e+03	8.87e+03
	Mean	8.97e+02	9.56e+02	2.75e+03	8.92e+03	8.09e+03
	Std	4.87e+01	1.44e+01	1.00e+03	2.52e+02	4.88e+02
		F16	F17	F18	F19	**F20**
	Best	1.70e-02	2.20e+03	2.48e+03	1.36e+00	2.35e+01
	Worst	1.62e-01	2.76e+03	2.69e+03	4.84e+00	2.50e+01
	Mean	5.39e-02	2.52e+03	2.52e+03	2.75e+00	2.49e+01
	Std	2.53e-02	1.25e+02	2.03e+02	8.42e-02	2.16e-01

Figures 28-47 show, for benchmark functions F1-F20, the best fitness achieved as CMA-ES progresses through the generations. These graphs can be represented as the fitness versus generation plots. From the plots, we can observe that the fitness for population gets better and better as the generations' progress, which exhibits the optimization property of the CMA-ES algorithm. For some functions the fitness improvement is very marginal based on the function, for such functions the fitness is plotted on a linear scale instead of logarithmic scale to show changes.

Similar graphs were captured for 30 and 50 Dimensions for all benchmark functions and exhibit similar results. Due to space constraints, these graphs are not listed in this document.

The performance of the CMA-ES algorithm is reported in Table 2, 3 and 4 on 10D, 30D and 50D benchmark functions, respectively. The values represented are the Best, Worst, Mean and standard deviations fitness values achieved before the exit criteria is reached over a series of 50 runs. Any value that is less than 1.0 e-08 is represented as 0.0 e +00.

A glance at the tables and graphs reveal that, optimal values are achieved within very few number of function evaluations (generations) varying from a minimum of 200 to a maximum of 1400 generations. This provides us proof that CMA-ES is indeed a major step forward in the field of optimization. The performance of the CMA-ES marginally decreased as the dimensionality of the search space increases. From the fact that the search space increases exponentially with the increase in dimensions (also known as curse of dimensionality [11]) continuous improvements to the optimization algorithms are needed to deal with these problem where highest level of optimization is desired.

6. CONCLUSION AND FUTURE WORK

As the requirement of real world optimization problems increase so does the demand for highly efficient and robust optimization algorithms. This research work evaluated the validity and performance of one such algorithm, Covariance Matrix Adaption Evolution Strategy Algorithm (CMA-ES) inspired by the biological evolution process, implemented using the Eclipse environment and Java 1.7 Version, on the first 20 benchmark functions specified in IEEE CEC' 2013 Test Suite.

The performance of the algorithm is evaluated by finding the best, worst, mean and standard deviations of fitness values achieved from the results of 50 runs for every objective function for 10D, 30D and 50D dimensions. The objective function fitness values over generations are tracked and plotted for all 20 Benchmark Functions with 10 dimensions. As expected, the results confirm that the algorithm efficiently solves the optimization problems in a lesser number of generations.

The algorithm has achieved best possible solution, however, there is still a scope of improvement. There are a number of enhancement approaches that have been worked on and still there is scope for improvement such as the Modified Covariance Matrix Adaption [10]. This can be achieved by varying the control parameters, Selection and Recombination approaches for generating the offspring for next generations. In the future, we plan to work on improving this process and validate if better results can be achieved with the update to the Covariance Matrix Adaption Evolution Strategies.

7. REFERENCES

[1] G. W. Greenwood, Finding Solutions to NP Problems, Published in proceedings CEC 2001, 815-822, 2001.

[2] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. Natural Computing 1(1), 3–52, 2002.

[3] N. Hansen,A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Proceedings of the 1996 IEEE Conference on Evolutionary Computation (ICEC ’96), pages 312–317, 1996.

[4] C. Igel, , T. Suttorp, N. Hansen: A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006), ACM Press 453–460, 2006.

[5] N. Hansen and A. Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The (μ/μI, λ)-ES. In EUFIT'97, 5th Europ.Congr.on Intelligent Techniques and Soft Computing, Proceedings, pp. 650-654, 1997.

[6] N. Hansen, The CMA Evolution Strategy: A Tutorial Nikolaus Hansen 2011.

[7] N. Hansen, The CMA Evolution Strategy [Online] https://www.lri.fr/~hansen/ cmaesintro.html .

[8] N. Hansen, SD. Muller, P. Koumoutsakos. Reducing the time complexity of the de-randomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation, 11(1):1–18, 2003.

[9] 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 2012, 2013.

[10] R. Chocat, L. Brevault Modified Covariance Matrix Adaptation – Evolution Strategy algorithm for constrained optimization under uncertainty, application to rocket design. IFMA, EA3867, Laboratoires de Mécanique et Ingénieries, Clermont Université, CP 104488, 63000 Clermont-Ferrand, France, 2015.

[11] R. E. Bellman, Dynamic Programming, ser. Dover Books on Mathematics. Princeton, NJ, USA: Princeton University Press / Mineola, NY, USA: Dover Publications, 1957/2003.

[12] R. Burger, The Mathematical Theory of Selection, Recombination, and Mutation. John Wiley & Sons, Chichester 2000.