**Indonesian Journal of Electrical Engineering and Computer Science**
Vol. 12, No. 1, October 2018, pp. 87~94
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i1.pp87-94


# Theoretical Analysis and Empirical Comparison of Different Population Initialization Techniques for Evolutionary Algorithms

**Devika. K, G. Jeyakumar**
Department of Computer Science and Engineering, Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham, India

**Article Info**

***Article history:***

Received Dec 26, 2017
Revised Jan 09, 2018
Accepted May 26, 2018

***Keywords:***

Differential Evolution
Evolutionary Algorithms
Population Initialization
Random Initialization and
Oppositional based
initialization

**ABSTRACT**

Evolutionary Algorithms (EAs) are the potential tools for solving optimization problems. The EAs are the population based algorithms and they search for the optimal solution(s) from a initial set of candidates solutions known as population. This population is to be initialized at first before the evolution of the algorithm starts. There exists different ways to initialize this population. Understanding and choosing the right population initialization technique for the given problem is a difficult task for the researchers and problem solvers. To alleviate this issue, this paper is framed with two objectives. The first objective is to present the details of various Population Initialization (PI) techniques of EAs, for the readers to give brief description of all the PI techniques. The second objective is to present the steps and empirical comparison of the results of two different PI techniques implemented for Differential Evolution (DE) algorithm. Theoretical insights and empirical results of the PI techniques are presented in this paper.



***Corresponding Author:***

Devika.K,
Department of Computer Science and Engineering,
Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
Email: cb.en.p2cse16008@cb.students.amrita.edu

## 1. INTRODUCTION

An optimization problem is a problem to which the best possible optimal solution is to be searched from a set of feasible ones in the population space. There exist numerous optimization algorithms in Mathematics and Computer Science. The Evolutionary Computation (*EC*) field of Computer Science has a set of algorithms known as Evolutionary Algorithms (*EAs*) for solving optimization problems. They are the most widely used tools to solve real time optimization problems [1-3]. It is a family of algorithms used for global optimization inspired from Darwin's theory of natural selection. Natural selection is the process by which the fittest candidate will survive longer. *EAs* are population based optimization algorithms. In *EAs*, a population of candidate solutions is generated initially and new solutions are generated iteratively by following the evolutionary process. Instances of *EAs* are Evolution Strategies (*ES*), Genetic Programming (*GP*), Evolutionary programming (*EP*), Genetic Algorithm (*GA*) and Differential Evolution (*DE*). In which, *DE* is a recent addition to *EA* family. All these instances follow the generate-and-test strategy of problem solving [4].

In *EA*, the population is a set of possible solutions for a given problem. Each solution is represented as a vector (also termed as chromosome). Each chromosome consists of attributes and each attributes hold some value. The evolutionary search of an *EA* starts with an Initial Population (*IP*) of solutions. In general, the *IP* is generated randomly if no problem specific information is known. Otherwise, problem specific heuristics can be added for the population initialization. Considering these facts, there exist different

population initialization (*PI*) techniques in the literature of *EC* community. This includes algorithm specific and problem specific *PI* techniques.

Each *PI* technique has its own characteristics. The objective of this paper is to present, in detail, different *PI* techniques of *EAs*. Further, in order to provide experimental evidences to the reader, this paper also discusses the results obtained by implementing two different *PI* techniques applied for *DE* algorithm. The theoretical information about *PI* techniques and the empirical results obtained using *DE* algorithm provided in this paper would definitely help the researchers in *EC* community to understand the importance of *PI* for solving the given problem.

The paper is organized as follows. Section 2 discusses about the framework of Evolutionary Algorithm, Section 3 introduces the population concepts of *EAs* and Section 4 presents the details of *PI* techniques. The Section 5 explains the design of experiment and the Section 6 presents the empirical results of *PI* techniques on *DE* algorithm. Finally, the Section 7 concludes the papers.

## 2. EVOLUTIONARY ALGORITHMS

To solve an optimization problem, a set of few possible solutions (candidates) to the problem is created initially and given as input to the *EA*. The nature of *EA* is to generate new candidates (children) from the selected candidates (parents) of the initial population, and allow the fittest candidates among the parents and children to survive for the next generation. This phenomenon is derived from the '*survival of the fittest*' concept of Darwin's natural evolution theory. The candidates in a population are supplied with limited resources in an environment. Under the environment pressure, the individuals must compete each other for the resources which results in survival of the fitter individuals. Based on the objective of the chosen problem, a fitness function is defined to measure the fitness of each candidate in the population. Based on the fitness values fittest candidates are chosen for the next generation. The evolutionary operators used during the *EA* process are Recombination and mutation. Recombination is applied between two or more selected candidates, which are called as parent candidates. Recombination results in one or more new candidates. Mutation is applied on a single candidate and results a new candidate. Mutation and recombination on selected candidates leads to the creation of new candidates (offsprings). Hence they are named as variation operators. Then the selection operator decides the survivors for next generation from the pool of parents and offsprings. The variation and selection operations are carried out iteratively until the algorithm reaches a user defined stopping criteria. It is very well evident that, after every iteration, the best candidate in the population moves towards the global optimal solution [4]. The general structure of *EA* is shown in Figure 1. As it is noted from the algorithmic structure shown in Figure 1, the *EA* comes under the category of generate-and-test algorithms. The most important components of *EA* are 1) Candidate Representation 2) Population Initialization 3) Fitness function 4) Parent selection mechanism 5) Variation operators and 6)Survivor selection mechanism

## 3. POPULATION

Population creates the unit of evolution for the *EA*. The candidates (individuals or chromosomes) in a population represent possible solutions of the problem to be solved. The number of parameters represented in a candidate is equal to the dimension of the problem. This is denoted as chromosome length. All the candidates in a population will have equal length. It is necessary to initialize/define a population by specifying the number of individuals (population size) in the population. Most of the cases the population size of the *EA* are kept constant. This leads to the competition for limited resources between the candidates. For instances, the fittest candidate of a population is selected for next generation and the worst candidate is replaced by the best candidate. The number of different candidates present in a population specifies the diversity of the population. The population with set of possible solutions is also termed as solution space. The dimension of the solution space is the dimension of the problem to be solved. The *EA*, start the search of global optimal solution from the initial population. As the search proceeds, the evolutionary operators (selection, recombination and mutation) bring changes in the population by adding/deleting/modifying the candidates. This evolutionary change in the population spaces can be depicted as an adaptive population landscape. An example adaptive landscape of a population with two dimensional problems is shown in Figure 2 (This figure is directly taken from [4]).

```
BEGIN
    Population Initialization
    Fitness Evaluation
    Repeat Until (Stopping Criteria is satisfied)
    DO
        1  Select Parents
        2  Perform Recombination between parents;
        3  Perform Mutation on the resulting off springs;
        4  Fitness Evaluation of new candidates;
        5  Select individuals for the next generation;
    LOOP
END
```

Figure 1. The structure of *EA*

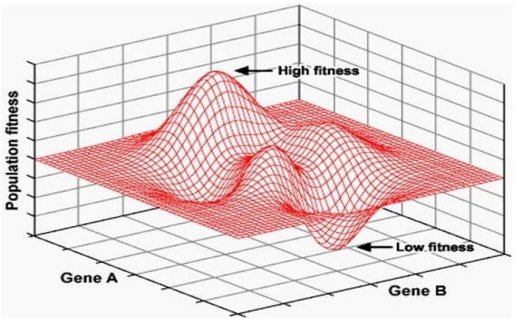


Figure 2. A sample landscape of a population

## 4. POPULATION INITIALIZATION TECHNIQUES

The population initialization is the initial step for all *EAs* and it randomly or heuristically provides initial guess of solutions. If the initial guesses are good, it helps the *EA* to find the optimal solution faster. The wrong initial guesses will affect the performance the *EA*, by directing it to waste the time in searching for the solutions in the non-promising area of the solution space. Also, it is difficult to determine whether the initialized population is good or not.

Since success of an *EA* starts with a good initial population, investigating and proposing different population initialization techniques is a good research area for the *EC* community. There are many population initialization (*PI*) techniques proposed by different researchers, which are supporting *EAs* in finding the optimal solution with less computational cost [5-8].

This section discusses different *PI* techniques commonly used in *EA* design. The different population initialization techniques are: Pseudo Random Number Generators (*PRNGs*), Chaotic Number Generators (*CNGs*), Quasi Random Sequence (*QRS*), Uniform Experimental Design (*UED*), Sobol Set (*SBL*), Good Lattice Point (*GLP*), Random Start Quasi Random Sequence (*RSQRS*), Scrambled Quasi Random Sequence (*SQRS*),Mixed Pseudo Random Sequence (*MPRS*), Oppositional Based Learning (*OBL*) and Centroid Voronoi tessellation (*CVT*)

These *PI* techniques can be categorized in to three groups, based on their characteristics and the way it works for generating the random numbers. The three broad categories are: Group 1: '*Randomness*' group, Group 2: '*Compositionality*' group and Group 3: '*Generality'* group

The *PI* techniques in which the random numbers are uniformly distributed and predictions of the future values are not possible are grouped under '*randomness'* group. The '*compositionality'* of *PI* techniques deals with the number of steps used for generating the population and '*generality'* deals with the usage of *PI* technique to solve normal problems as well as to solve specific problems. Each group has again two sub categories. A chart visualizing this categorization is shown in Figure 3, and the sub categories are [5] *Stochastic* - Population depends upon the initial seed, *Deterministic* - Always generate same population, *Non-compositional*:- Produces population in a single step, *Compositional*: - Comprises more than one step, *Generic*: - Can be used in all type of optimization problems and *Application specific*: - Applicable to particular real world problems.

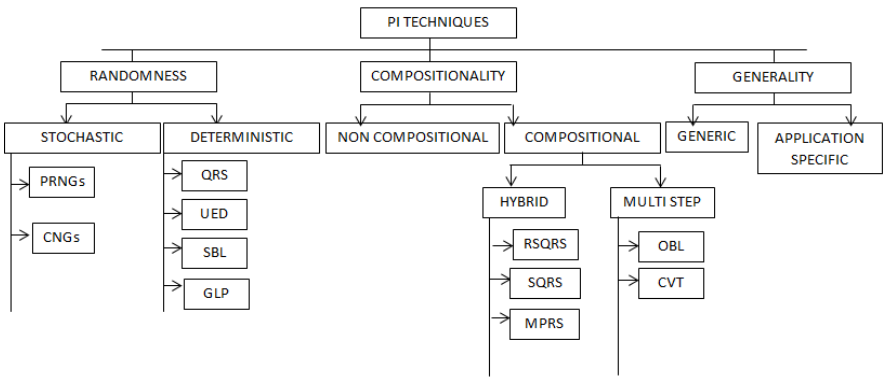


Figure 3. Categorization of PI techniques

### 4.1. Pseudo Random Number Generators (*PRNGs*)

As computers fails in producing true random numbers, pseudo random numbers are used to generate random numbers. *PRNGs* will generate numbers which look like random. Ranking of different *PRNGs* can be done based on two factors – *Cycle_Time* and *Equidistribution*.

*Cycle_Time* is the smallest integer that the *PRNG* repeats producing from the previously produced numbers and *equidistribution* is the range of points which have equal distribution [5]. The different pseudo Random generators are: (1) *GCC RAND (2)* Multiply With Carry Generator (*MWC*) (3) Complimentary Multiply With Carry Generator (*CMWC*) (4) Linear Congruential Generator (*LCG*) (5) XOR Shift Generator (*XOR*) (6) Mersenne Twister (*MT*) (7) Well Equidistributed Long Period Linear (*WELL*) (8) Keep It Simple Stupid (*KISS*).

The *GCC RAND* is an inbuilt *PRNG* which is available in all programming languages like C, C++. The *MWC* can be used to generate random numbers quickly and slight modification in modulo arithmetic give rise to another *PRNG* called *CMWC*. The *XOR PRNG uses Exclusive-OR* Boolean operation to generate random numbers. One of the well-known and the oldest *PRNG* is *LCG* and it generates non continuous random numbers using linear equations. To generate high quality random numbers *MT* is used and it's the most widely used *PRNG*. Mersenne Twister name is derived from Mersenne Prime because the period length chosen to be a Mersenne Prime. Mersenne Prime is a prime number that is one less than power of two. The *MT* is the first *PRNG* to generate fast and high quality random numbers. Commonly used version of *MT* algorithm is based on Mersenne Prime $2^{19937}$-1. Different versions of *WELL* generator are proposed for generating random numbers. The *KISS PRNG* can't be used in context where cryptographic security is important. The *MT* is available in all programming languages so it's the most popular method in *PRNG*. Choice of *PRNG* can affect the performance of *EA* [5, 9, 10]. The *PRNG* has several merits and demerits. The merits are:

a. PRNG's are easily available in every programming language.
b. There is no restriction for population size and on the number of decision variables.
c. Simple technique.
d. Uniform population can be generated
e. Transformation from uniform population to biased population is easy.

The demerits are:

a. Can't generate perfect evenly distributed points.
b. PRNG suffers from the curse of dimensionality.

These demerits will affect EA process more when the search space is vast and the dimensionality of the problem is too low.

### 4.2. Chaotic Number Generators (*CNGs*)

The working of *CNGs* is based on chaos theory. Chaos is very sensitive to initial conditions. It is difficult to predict the numbers generated by *CNGs*. The *CNGs* mainly use recursive algorithms. To generate chaotic sequence an initial seed is selected randomly and a function (map) is applied on it. The map is applied several times to the previously generated numbers to get the sequence. Different types of one dimensional and two dimensional maps are available [5, 11]. They are Circle Map, Cubic Map, Gauss Map, ICMIC map, Logistic Map, Sinusoidal Iterator, Tent Map, Baker's Map, Arnold's Map and Zaslavskli's Map

Baker's, Arnold's and Zaslavskli's are two dimensional maps and all others are one dimensional maps. To generate a population using *CNG* proper maps are required. Tent map is most commonly used in *CNG*, because it has higher iterative speed compared to other maps. Tent map generate uniformly generated chaotic sequence within the range of [0, 1].

The main properties of *CNG* are ergodicity, randomness and regularity. Opting *CNG* as *PI* technique will improve the performance of *EA* in terms of population size, success rate and convergence rate [5, 11].

### 4.3. Quasi Random Sequence (*QRS*)

The *QRS* will generate the sequences which are neither true random nor pseudo random and it doesn't require any random element. The *QRS* is also called as low discrepancy sequences or sub random sequences. In worst case *QRS* can be non-uniform. Compared to *PRNG*, the *QRS* is used in high dimensional problems. Sometimes the numerical algorithms in the *QRS* will contradict each other [5].

### 4.4. Uniform Experimental Design (*UED*)

It is a type of space filling algorithm which looks for the points that have to be evenly distributed in a given range. The *UED* is mainly used in computer simulated designs. The *QRS* uses only one dimension projection whereas *UED* uses *D* dimension projections this is one advantage of *UED* over *QRS* [5].

### 4.5. Sobol Set (*SBL*)

The *SBL* will generate populations which are well distributed in the decision space. The Sobol sequence values will be in between zero and one. Most commonly used algorithm for generating sobol sequence is *Algorithm-659* and it can generate the integers up to 40 dimensions [8].

### 4.6. Good Lattice Point (*GLP*)

The *GLP* generates points which are evenly distributes in the decision space. Lattice point is a group of points in the same location. Lattice rules are used to create sequence in *GLP* [8].

### 4.7. Centroid Voronoi Tessellation (*CVT*)

The *CVT* helps to divide the search space in equal volumes. The *CVT* doesn't use fitness function to evaluate the population. Initial population is created using any of the *PI* techniques. The population space is divided into some partitions using randomly generated auxiliary points. These partitions are iteratively enhanced till the termination criteria are met [5].

### 4.8. Oppositional Based Learning (*OBL*)

Initially, *OBL* generates population called original population. The original population can be generated using any of the existing population initialization techniques. Then a new population is created by applying some heuristics rules and that new population is called opposite population. By comparing the fitness of the candidates in both the populations, a subset is created from the union of initial and opposite populations. Main goal of *OBL* is to make the population closer to the optimal solution [5, 8]. The variants of *OBL* are Quasi Opposition Based Learning (*QOBL*), Quasi Reflection Opposition Based Learning (*QROBL*), Center Based Sampling (*CBS*), Generalized Opposition Based Learning (*GOBL*) and Current Optimum Opposition Based Learning (*COOBL*)

In *QOBL* instead of actual opposite point quasi opposite point is used. Quasi opposite point is a point which is randomly generated and it's located between the opposite point and the middle point. The performance of *OBL* and its variants depends upon the original population. Compared to *OBL* and *QOBL* the *QROBL* will generate the population which is more close to the optimal solution. The *OBL* tends to find the optimal solution faster than other *PI* techniques. Several studies claim that for wide range of problems the best performing *PI* technique is *OBL* [10]. The *CNG*, *OBL* and *QOBL PI* techniques can work well on both high dimensional and low dimensional problems. Studies show that *PI* component of *EA* is an interesting research segment for *EC* community. Further research studies on this can add new advanced *PI* techniques to work well on large scale optimization problems.

As discussed above, there exist several *PI* techniques for *EA*. The performance of the *EA* can be affected by the usage of particular *PI* technique. Categorization of *PI* techniques gives a rough picture about the working of various *PI* techniques. Each technique has different variants. The *PI* is the initial stage of all *EA*, therefore it is important to choose the best suitable *PI* technique for the problem to be solved. The studies on *PI* techniques reveal that *OBL* is the best *PI* technique for *EA* [12, 13] which helps in generating high quality solutions. The *OBL* based *PI* was experimented on *DE* algorithm in [14].

## 5. DESIGN OF EXPERIMENT

The *DE* algorithm proposed by Rainer Storn in [15] is used in this experiment. *DE* has the algorithmic structure similar to other *EAs* [16], however it has a unique mutation scheme called as *differential mutation*. There are many research works to propose improved *DE* algorithms [17, 18] and to use it for real time optimization problems [19]. There exists few works where different population initialization is experimented for *DE* [20, 21]. A research work to study the performance of *OBL PI* technique using convergence speed is presented in [12], for a *DE* variant with 34 benchmark functions. The population dynamics of the chosen *DE* algorithms is analyzed well and reported in the literature using the random *PI* technique [22-25].

The empirical results obtained by implementing two *PI* techniques for *DE* is presented in this Section. The *DE* algorithm has been implemented using random and *OBL PI* techniques. The experimental setup includes 4 different *DE* variants and 4 benchmarking functions. The *DE* variants used are *DE/rand/1/bin*, *DE/rand/1/exp*, *DE/best/1/bin* and *DE/best/1/exp*. The benchmark functions used are Sphere model – $f_1$, Schewefel's problem - $f_2$, Generalized Rosenbrock's function – $f_3$ and Generalized Schewefel's problem – $f_4$. The details of the benchmarking functions are presented in Table 1. The design experiment includes setting up values for the parameters. The parameters for *DE* algorithm and their corresponding values used in the experiment are shown in Table 2.

The *PI* techniques used in the experiment are random *PI* and *OBL PI*. The *DE* algorithms with these *PI* techniques are names as $DE_{RPI}$ and $DE_{OBLPI}$, henceforth. They are implemented for two different population sizes: *NP* = 5 (with *D* = 5) and *NP* = 60 (with *D* = 30). In $DE_{RPI}$, the values for each of the component of a chromosome are generated randomly within the allowed range mentioned in the benchmarking functions. For the chromosome *i* of the population *X*, the $j^{th}$ component is initialized as follows

$$X_i[j] = xl + (xu - xl) * rand(seed) \quad (1)$$

where *xl* and *xu* are the lower and upper bound of the allowed values of the component and *seed* is the input for the random number generator.

In $DE_{OBLPI}$, an initial population is created randomly (as above) then an opposite population is generated with this initial population which contains the opposite of each individual. The opposite candidate ($OX_i$) for each candidate ($X_i$) in the population is created using the equation (2). New population is created by combining the initial population and opposite population. Then the best *NP* candidates from the combined population are selected for initial population.

$$OX_i[j] = xl + xu - X_i[j] \quad (2)$$

The performance of $DE_{RPI}$ and $DE_{OBLPI}$ is compared by the mean objective function (*MOV*) values. The *MOV* is the average of the best objective function values obtained by the algorithm at the end of each run. It is calculated as follows

$$MOV = (\sum_{i=1}^{MaxRun} BestOv_i)/MaxRun \quad (3)$$

where *MaxRun* is the maximum number of runs (which is set as 50) and $BestOV_i$ is the best objective function value obtained by the algorithm for the run *i*.

Table 1. Functional description of the benchmarking functions

| **$f_1$ – Sphere model** | **$f_2$ – Schwefel's Problem 1.2** |
|---|---|
| $f_{sp}(x) = \sum_{i=1}^{n} x_i^2$<br>$-100 \le x_i \le 100;\ x^* = (0,0,\dots,0);$<br>$f_{sp}(x^*) = 0;$ | $f_{sch}(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$<br>$-100 \le x_i \le 100;\ x^* = (0,0,\dots,0);$<br>$f_{sch}(x^*) = 0;$ |
| **$f_3$ - Generalized Rosenbrock's Function** | **$f_4$ – Generalized Schwefel's Problem 2.26** |
| $f_{gr}(x) = \sum_{i=1}^{n} \lvert 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \rvert$<br>$-30 \le x_i \le 30;$<br>$x^* = (0,0,\dots,0);$<br>$f_{gr}(x^*) = 0;$ | $f_{gsch}(x) = \sum_{i=1}^{n}\left(x_i sin\left(\sqrt{\lvert x_i \rvert}\right)\right)$<br>$-500 \le x_i \le 500;$<br>$x^* = (420.9678,\dots,420.9678);$<br>$f_{gsch}(x^*) = -12569.486618164879;$ |

Table 2. The parameter set up for the experiment

| Sno | Parameter | Value |
|---|---|---|
| 1 | Population Size (NP) | 5 and 60 |
| 2 | Dimension (D) | 5 and 30 |
| 3 | Crossover Rate ($C_r$) | 0.9 |
| 4 | Mutation Step Size (F) | 0.1 to 0.9 |
| 5 | Maximum Number of Generation (*MaxGen*) | 1000 |
| 6 | Number of runs (*MaxRun*) | 50 |

## 6. RESULTS AND DISCUSSIONS

The chosen benchmarking problems are solved by the $DE_{RPI}$ and $DE_{OBLPI}$ algorithms for two different parameter set ups considering for smaller and larger population sizes (as mentioned below).

(1) *NP* = 5, *D* = 5, $C_r$ = 0.9, *F* = 0.1 to 0.9, *MaxGen* = 1000 and *MaxRun* = 50 and

(2) *NP* = 60, *D* = 30, $C_r$ = 0.9, *F* = 0.1 to 0.9, *MaxGen* = 1000 and *MaxRun* = 50

Each benchmarking functions are solved by four variants of both the algorithms. Considering 4 benchmarking functions, 4 Variants and 2 algorithms, the experiment covers 32 possible combinations of functions, algorithms and the variants.The *MOVs* measured for the smaller population size (ie., *NP* = *5*) is presented in Table 3, and for the larger population size (ie *NP* = 60) it is presented in Table 4.

It is observed from the comparative results that for the lower population size (Table 3) the $DE_{OBLPI}$ outperforms $DE_{RPI}$ only in 7 out of 16 cases, by providing more accurate solutions. However for the larger population size (Table 4) the $DE_{OBLPI}$ outperforms $DE_{RPI}$ in all the cases, except in the case of $f_1$-*rand/1/bin* combination where both the algorithms performed similar. It is worth noting that the effect opposite population in population initialization is not significant while the population size is smaller. In case of larger population size the opposite population had greater influence in the initial population thus all the function and variant combinations were able to achieve good solutions. This proves the novelty of the *OBL PI* technique in improving the search of *EA*. Thus the validity of the *OBL PI* is reiterated in this small experimental set up.

Table 3. *MOV* obtained for *NP* = 5

| Function | Variant | $DE_{RPI}$ | $DE_{OBLPI}$ |
|---|---|---|---|
| $f_1$ | *rand/1/bin* | 1284.85 | 2022.57 |
| | *rand/1/exp* | 1105.60 | 1400.32 |
| | *best/1/bin* | 2582.08 | 10216.14 |
| | *best/1/exp* | 1956.78 | 1174.14 |
| $f_2$ | *rand/1/bin* | 1167.37 | 1815.35 |
| | *rand/1/exp* | 1233.14 | 5201.73 |
| | *best/1/bin* | 2392.33 | 661.05 |
| | *best/1/exp* | 626.10 | 5293.89 |
| $f_3$ | *rand/1/bin* | 467173 | 93637.02 |
| | *rand/1/exp* | 43856.51 | 77850.57 |
| | *best/1/bin* | 1542797 | 2564800 |
| | *best/1/exp* | 442737.31 | 179507.71 |
| $f_4$ | *rand/1/bin* | 5664.98 | 7721.12 |
| | *rand/1/exp* | 5706.62 | 5482.35 |
| | *best/1/bin* | 10817.67 | 10187.12 |
| | *best/1/exp* | 10230.97 | 9114.67 |

Table 4. *MOV* obtained for *NP* = 5

| Function | Variant | $DE_{RPI}$ | $DE_{OBLPI}$ |
|---|---|---|---|
| $f_1$ | *rand/1/bin* | 0.00 | 0.00 |
| | *rand/1/exp* | 1.78 | 1.45 |
| | *best/1/bin* | 3284.47 | 3183.07 |
| | *best/1/exp* | 734.18 | 640.93 |
| $f_2$ | *rand/1/bin* | 46.65 | 24.63 |
| | *rand/1/exp* | 475.83 | 445.17 |
| | *best/1/bin* | 5162.91 | 3700.08 |
| | *best/1/exp* | 25.99 | 16.83 |
| $f_3$ | *rand/1/bin* | 40.68 | 35.10 |
| | *rand/1/exp* | 731.35 | 434.53 |
| | *best/1/bin* | 1370607 | 1137933 |
| | *best/1/exp* | 463698.81 | 319656 |
| $f_4$ | *rand/1/bin* | 0.15 | 0.08 |
| | *rand/1/exp* | 0.09 | 0.03 |
| | *best/1/bin* | 130.26 | 0.12 |
| | *best/1/exp* | 0.01 | 0.00 |

## 7. CONCLUSIONS

This paper presents a survey on existing *PI* techniques for *EAs*. The categorization presented in the paper would support the researchers to get an insight of various *PI* techniques and their types. This in turn will help them to select a suitable *PI* technique for solving their problem. The experimental details of implementing the random *PI* and *OBL PI* techniques for *DE* algorithm also presented in this paper. The comparative study of these two techniques reveals that the *OBL PI* technique is performing better than the random *PI* technique for larger population sizes.

As an initial attempt, this study is done on a simple experimental setup with one 4 *DE* variants and 4 benchmarking functions. However, this work can be extended further by implementing all the *PI* techniques on a single *EA* with uniform design of experiment. A systematic empirical comparison of all the *PI* techniques can be performed based on the results, to understand and validate the unique nature of each of them.

## REFERENCES

[1] Sandip Chanda, Abhinandan De, "Congestion Relief of Contingent Power Network with Evolutionary Optimization Algorithm". *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, Vol.10, No.1, pp. 1-8, 2012.

[2] Shufang Wu, Tiexiong Su, "Optimization Design of Cantilever Beam for Cantilever Crane Based on Improved GA", *Indonesian Journal of Electrical Engineering and Computer Science*, Vol.12, No.4, pp. 2652 - 2657, 2014.

[3] Liu Xiaoxiong, Wang Juan, Wu yan, Liu Yu, "The Optimization of Lateral Control Augmentation based on Genetic Algorithms". *Indonesian Journal of Electrical Engineering and Computer Science***,** Vol. 11, No. 6, pp. 2962 – 2967, 2013.

[4] Eiben, Agoston E., and James E. Smith. *Introduction to evolutionary computing*. Vol. 53. Heidelberg: springer, 2003.

[5] Kazimipour Borhan, Xiaodong Li, and A. Kai Qin. "A review of population initialization techniques for evolutionary algorithms." *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014.

[6] Kondamadugula, Sita, and Srinath R. Naidu. "Accelerated evolutionary algorithms with parameter importance based population initialization for variation-aware analog yield optimization." *Circuits and Systems (MWSCAS), 2016 IEEE 59th International Midwest Symposium on*. IEEE, 2016.

[7] Kazimipour, Borhan, Xiaodong Li, and A. Kai Qin. "Initialization methods for large scale global optimization." *2013 IEEE Congress on Evolutionary Computation (CEC)*, 2013.

[8] Kazimipour Borhan, Xiaodong Li, and A. Kai Qin. "Effects of population initialization on differential evolution for large scale optimization." *2014 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2014.

[9] Rajashekharan, Lekshmi, and C. Shunmuga Velayutham. "Is Differential Evolution Sensitive to Pseudo Random Number Generator Quality?–An Investigation." *Intelligent Systems Technologies and Applications*. Springer, Cham, 2016. 305-313.
[10] Segredo, Eduardo, et al. "On the comparison of initialisation strategies in differential evolution for large scale optimisation." *Optimization Letters* , pp. 1-14, 2017.
[11] Lu, Hui, et al. "The effects of using Chaotic map on improving the performance of multiobjective evolutionary algorithms." *Mathematical Problems in Engineering* 2014 (2014).
[12] Rahnamayan, Shahryar, Hamid R. Tizhoosh, and Magdy MA Salama. "A novel population initialization method for accelerating evolutionary algorithms." *Computers & Mathematics with Applications*, Vol. 53, No. 10, pp. 1605-1614, 2007.
[13] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M. A. Salama, "Opposition-Based Differential Evolution". *IEEE Transactions On Evolutionary Computation*, Vol. 12, No. 1, 2008.
[14] Rahnamayan S., Tizhoosh H.R. "Differential Evolution Via Exploiting Opposite Populations". In: Tizhoosh H.R., Ventresca M. (eds) *Oppositional Concepts in Computational Intelligence, Studies in Computational Intelligence*, Vol 155, pp 143-160. Springer, Berlin, Heidelberg, 2009.
[15] Storn, Rainer, and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces." *Journal of global optimization*, Vol. 11, No. 4, pp. 341-359, 1997.
[16] Jianfeng Qiu, Jiwen Wang, Dan Yang, Juanxie. "A Comparison of Improved Artificial Bee Colony Algorithms Based on Differential Evolution", *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 11, No. 10, pp. 5579 – 5587, 2013.
[17] Lingjuan HOU, Zhijiang HOU, "A Novel Discrete Differential Evolution Algorithm", *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 11, No. 4, pp. 1883~1888, 2013.
[18] Jeyakumar, G. and ShunmugaVelayutham, C. "Distributed Heterogeneous Mixing of Differential and Dynamic Differential Evolution Variants for Unconstrained Global Optimization", *Soft Computing – Springer*, Volume 18, Issue 10 (2014), Page 1949-1965, October -2014.
[19] Zain Zaharn, Ruifeng Shi, Xiangjie Liu, "The Power Unit Coordinated Control via Uniform Differential Evolution Algorithm". *Indonesian Journal of Electrical Engineering and Computer Science*, Vol.11, No. 7, pp. 3498 - 3507, 2013.
[20] Wang, Jiahai, Weiwei Zhang, and Jun Zhang. "Cooperative differential evolution with multiple populations for multiobjective optimization." *IEEE transactions on cybernetics* 46.12 (2016): 2848-2861.
[21] Salehinejad, Hojjat, and Shahryar Rahnamayan. "Effects of centralized population initialization in differential evolution." *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016.
[22] S. Thangavelu,G. Jeyakumar and C. Shunmuga Velyautham, "Population Variance Based Empirical Analysis of the Behavior of Differential Evolution Variants", *Applied Mathematical Sciences*, HIKARI Ltd, Vol. 9, No. 66, pp. 3249 – 3263, 2015.
[23] Ramya Raghu and G Jeyakumar, "Empirical Analysis on the Population Diversity of the Sub-populations in Distributed Differential Evolution Algorithm," *In Proceedings of Springer International Conference on Soft Computing Systems*, and in *International Journal of Control Theory and Applications*, Vol. 8,No. 5, pp. 1809-1816, 2016.
[24] M.S. Akhila, C.R. Vidhya and G. Jeyakumar, "Population Diversity Measurement Methods to Analyze the Behavior of Differential Evolution Algorithm," *In Proceedings of Springer International Conference on Soft Computing Systems*, and in *International Journal of Control Theory and Applications*, Vol. 8, No. 5, pp. 1709-1717, 2016.
[25] Ramya Raghu and G.Jeyakumar, "Mathematical Modelling of Migration Process to Measure Population Diversity of Distributed Evolutionary Algorithms", *Indian Journal of Science and Technology*, Vol, 9., No. 31, pp. 1-10, 2016.