This article can be cited as M. Kaedi, Fractal-based Algorithm: A New Metaheuristic Method for Continuous Optimization, International Journal of Artificial Intelligence, vol. 15, no. 1, pp. 76-92, 2017.
Copyright©2017 by CESER Publications

# Fractal-based Algorithm: A New Metaheuristic Method for Continuous Optimization

**Marjan Kaedi**[1]

[1]Faculty of Computer Engineering, University of Isfahan, Hezar-Jerib St.,
Isfahan 81746-73441, Iran
Email: kaedi@eng.ui.ac.ir

## ABSTRACT

*In this paper a population-based metaheuristic algorithm named fractal-based algorithm is developed to solve continuous optimization problems. In this algorithm, the density of high quality and promising points in an area is considered as a heuristic which estimates the degree of promise of that area for finding the optimal solution. Afterward, the promising areas of state space are iteratively detected and partitioned into self-similar and fractal-shaped subspaces for being searched more precisely and more extensively. The proposed algorithm is compared with some metaheuristic algorithms. The results demonstrate that the algorithm is able to find high quality solutions within appropriate time.*

**Keywords:** Continuous optimization, Heuristic, Promising subspace, Fractal-shaped structure.

**Mathematics Subject Classification**: 26Bxx, 68Wxx

**Computing Classification System:** I.2.8, F.1.2

## 1. INTRODUCTION

Metaheuristic algorithms have been highly popular in recent years, because many of today's optimization problems are large, complex and dynamic. Solution of such problems requires methods which can find acceptable solutions within a reasonable period of time rather than assure the finding of the optimal solution (Chiong, 2009; Yang, 2010).

Metaheuristic algorithms can be classified into different categories. One criterion for the classification of such methods is the number of candidate solutions evaluated in each iteration. Based on this criterion, they are divided into two general categories (Nicoară, 2012): single-solution-based metaheuristics and population-based metaheuristics.

Single-solution-based metaheuristics focus on a single solution during the search process, and attempt to improve that solution by introducing minor changes to it in an iterative process (Talbi, 2009; Spall, 2003; Zäpfel and Braune, 2010). Such searches can be thought of as walking in the problem state space (Talbi, 2009).

Population-based metaheuristics, on the other hand, work on a set of solutions known as population, and improve the population in an iterative process (Talbi, 2009; Spall, 2003). Many population-based metaheuristic algorithms have been developed, including genetic algorithms (Holland, 1975; Back, et al., 1997), estimation of distribution algorithms (Larranaga and Lozano, 2002; Pelikan, et al., 2006), scatter search (Glover, 1999; Glover, et al.,2000), ant colony optimization (Dorigo, 1999), particle swarm optimization (Kennedy and Eberhart, 1995; Kennedy and Eberhart, 2001), differential evolution (Storn and Price, 1997), honey bee algorithm (Nakrani and Tovey, 2004; Pham, et al., 2005; Karaboga, 2005), firefly algorithm (Yang, 2010), and cuckoo search algorithm (Yang and Deb, 2009). Such algorithms have proved to be highly efficient in solving many optimization problems (Ali, et. al., 2016; Ayan, et al., 2015; Azar, et. al., 2016; Castillo, et al., 2015; Chávez-Conde, et al., 2015; Chul, et al., 2015; Ghosn, et. al., 2016; Glotić and Zamuda, 2015; Gotmare, et al., 2015; Manikandan, 2014; Martí, et al., 2015; Qi, et al., 2015; Precup, et al., 2014; Raja, et al., 2015; Ramírez-Ortegón, et. al., 2013; Wang, et al, 2015). Some of the population-based metaheuristic algorithms have been adopted for solving the dynamic optimization problems (Kaedi, et al., 2013; Kaedi, et al., 2016), robust optimization problems (Moraes, et al., 2015), and multi objective optimization problems (Martin, et al., 2009).

This paper develops a new population-based metaheuristic algorithm to solve continuous optimization problems. The proposed algorithm, which we call ‘fractal-based algorithm’, partitions the promising areas of the state space of the continuous optimization problems in an iterative process and on the basis of self-similar and fractal-shaped structures, and in this way it attempts to seek the optimal solution.

Later on, in Section 2, the heuristic used in this paper is introduced. Afterwards, in Section 3, the proposed metaheuristic algorithm named fractal-based algorithm is introduced. In Section 4, the proposed algorithm is used to solve some continuous benchmark optimization problems and is compared with other population-based algorithms. Finally, in Section 5, some conclusions are drawn.

## 2. THE PROPOSED HEURISTIC

If we consider the state space of the continuous optimization problems, the quality of adjacent points in this space is changing and oscillating in a continuous and gradual way and in the neighbourhood of an optimal point (local or global optima), as we move toward the optimal point, the quality of the points in the state space is gradually improved (Figure 1). Therefore, the quality of neighbouring points is not independent of one another, and usually the presence of a number of high quality points close to one another can be a sign of presence of other high quality points in that neighbourhood and the strong probability of the presence of optimal points in that area. Hence, the density of high quality and promising points in an area in the state space can be considered as a heuristic which can predict the degree of promise of that area of the state space for finding the optimal solution.

It should be noted that this heuristic, like any other heuristic, does not assure the finding of the optimal solution; rather, it is merely a method based on conjecture to predict promising areas, and, like other heuristics, can have exceptions and in some cases may mislead the algorithm in the course of search. For instance, in cases where there is a flat area in the fitness landscape (Figure 1), the density of relatively high quality points in a neighbourhood is high, while there is no optimal point (neither local nor global) in that neighbourhood.

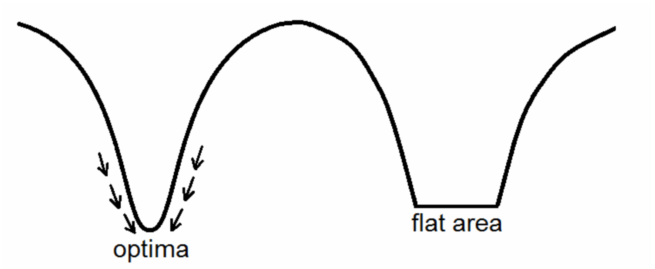


**Figure 1.** The optima and a flat area in a 2-dimnsional fitness landscape

This paper develops a metaheuristic algorithm for continuous optimization where the heuristic of density of high quality points in a neighbourhood is used to find the promising areas of the state space. This algorithm is introduced in Section 3.

## 3. THE PROPOSED ALGORITHM: FRACTAL-BASED ALGORITHM

Without loss of generality, we assume that the aim of the optimization is to find the minimum of a continuous function $f$. Certainly, by applying minor changes to the algorithm, one can generalize the proposed algorithm to find the maximum of a continuous function. As mentioned in Section 2, in this study, the subspaces of the state space containing larger numbers of random promising points are regarded as promising areas to find the optimal point. In the proposed algorithm, the following stages are executed iteratively:

First, the state space is divided into some subspaces with equal sizes. Next, some random points are generated in the state space uniformly, the value of the target function is calculated for the randomly generated points, and the promising points (i.e., points with lower values of $f$ ) are chosen. Afterwards, the number of promising points which have fallen into each subspace of the state space is determined. Based on the heuristic introduced in Section 2, the subspaces containing more promising points are considered as the promising subspaces, where the chance of finding the optimal solution is higher. Later on, these subspaces are in turn divided into smaller subspaces and the entire process of the algorithm is repeated on those subspaces so that they are searched more precisely and more extensively. The details of the proposed algorithm are as follows:

1. It is assumed that the goal is to find the minimum of a continuous function in an *n*-dimensional space where. The problem is defined as follows:

$$Minimize\ f(x_1, x_2, ..., x_n) \qquad where \quad L_i \le x_i \le U_i \quad for\ \ 1 \le i \le n \tag{1}$$

Therefore, the points for which the value of function $f$ is smaller are regarded as the more promising points. At first the entire state space is considered as the promising area.

2. The entire state space is divided into a number of subspaces. For this purpose, the $d^{th}$ dimension of the state space ($1 \le d \le n$) is divided into as many as $m_d$ equal subintervals until finally a grid containing $m_1 \times m_2 \times ... \times m_n$ subsections is built. Thus, we have:

Size of each subinterval in $d^{th}$ dimension $= \dfrac{U_d - L_d}{m_d}$ (2)

3. A uniformly distributed initial population is generated randomly all over the promising area (i.e., all over the state space). This population is considered as the current population.

4. The value of function $f$ for each point of the current population is calculated.

*5.* $P_1$ percent of *the* points of the current population with the lowest values of function $f$ are regarded as the promising points.

6. The number of promising points in each of the subspaces is determined. This number is an indication of the degree of promise of that subspace (i.e., the subspace promising rank).

$PromisingRank_s$= Number of promising points in $s$ (3)

*7.* $P_2$ percent of the most promising subspaces are selected to be searched more precisely and more extensively.

7.1. More precise search in the promising subspaces: The promising subspaces are in turn divided into smaller subspaces. For this purpose, the $d^{th}$ dimension ($1 \le d \le n$) of each subspace is divided into as many as $m_d$ equal intervals so that a grid containing $m_1 \times m_2 \times ... \times m_n$ subsections is built within that subspace. In this way, these promising subspaces will be searched more precisely because the algorithm will focus on the finer parts of them.

7.2. More extensive search in the promising subspaces: A number of random points are generated all over the state space. They are called the new population. In generating these random points, the points are not uniformly distributed all over the state space. Rather, more points are generated in the more promising subspaces so that those subspaces are searched more extensively. For this purpose, the number of points generated in each subspace is a linear function of the degree of promise of that subspace. Thus, for subspace $s$ we have:

Number of points generated in $s$ = ($PromisingRank_s$ / $\sum_{k \in all\ subspaces} PromisingRank_k$) × $PopulationSize$ (4)

8. The value of function $f$ is calculated for each point in the new population.

9. The fittest points in the new population and the current population are integrated based on the truncation selection method (Holland, 1975; Back, et al., 1997) and a population equal in size to the current population is built to replace the current population.

*10.* $P_3$ percent of the points in the new population are randomly selected and modified slightly by adding a *Gaussian* noise to them (like as mutation operation in genetic algorithm).

11. The termination condition is evaluated and in case the termination condition is not reached, the process is repeated from stage 5.

12. The best solution generated so far is returned as the output of algorithm.

Therefore, by iterating the algorithm, the promising areas of the state space are divided into smaller subspaces, and, the more promising subspaces of them are in turn divided into still smaller subspaces and so on. As a result, some grids similar to what divided the entire state space into subspaces are formed in the primary promising subspaces and then in the secondary promising subspaces, etc., so that the promising areas are identified more accurately and are focused upon during the search process. Consequently, some self-similar and fractal-shaped structures are formed in the state space, which are expected to direct the algorithm toward the optimal solution (Figure 2). The flowchart and pseudocode of the proposed algorithm are presented in Figures 3 and 4, respectively. It should be noted that the values of parameters $P_1$, $P_2$, and $P_3$ control the tradeoff between exploration and exploitation during the search and affect the algorithm convergence.

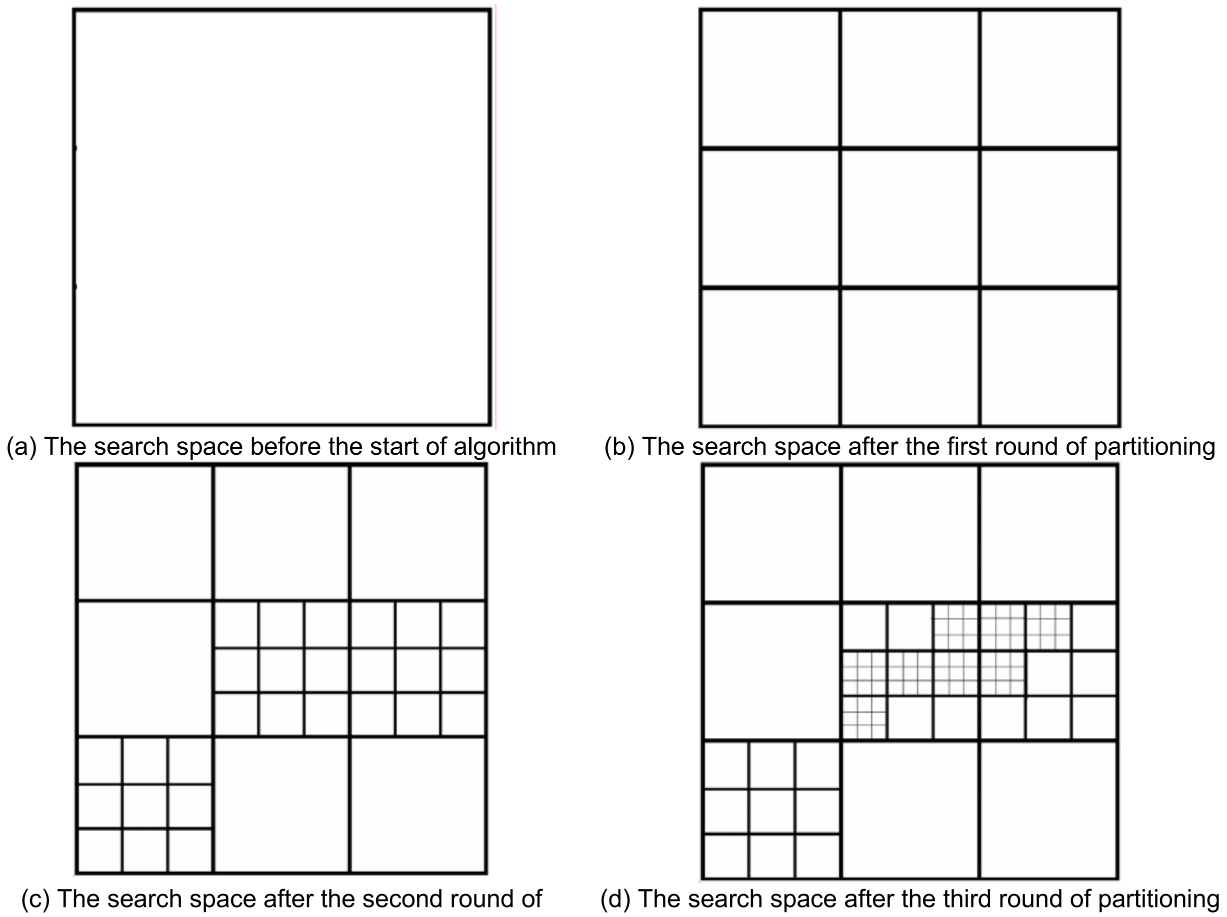

(a) The search space before the start of algorithm
(b) The search space after the first round of partitioning
(c) The search space after the second round of partitioning
(d) The search space after the third round of partitioning

**Figure 2.** Forming the self-similar and fractal-shaped structures in the state space during the run of the fractal-based algorithm (promising subspaces are gradually divided into smaller subspaces so as to be searched more precisely and more extensively).

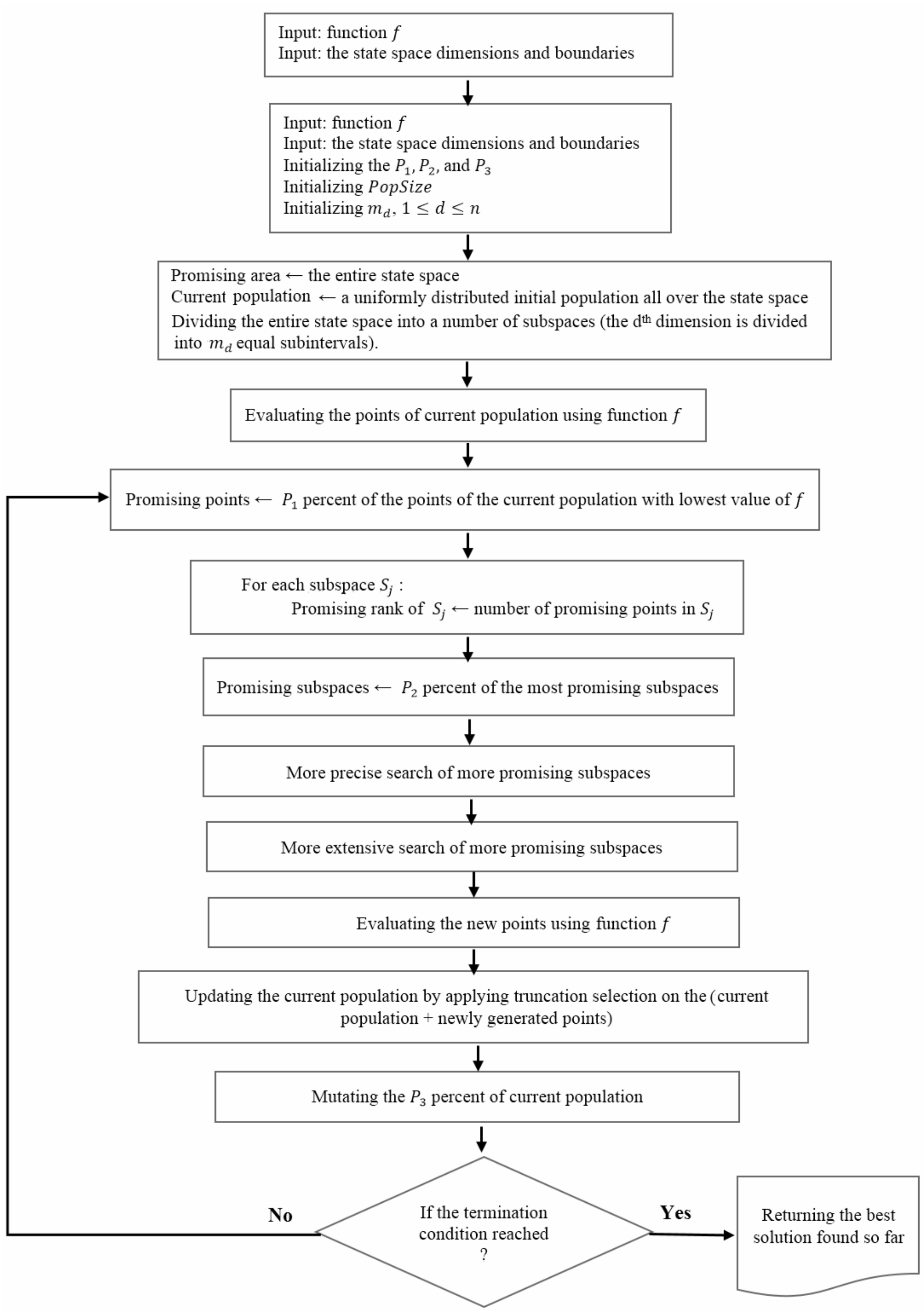

Input: function $f$
Input: the state space dimensions and boundaries
Input: function $f$
Input: the state space dimensions and boundaries
Initializing the $P_1, P_2$, and $P_3$
Initializing $PopSize$
Initializing $m_d$, $1 \leq d \leq n$
Promising area ← the entire state space
Current population ← a uniformly distributed initial population all over the state space
Dividing the entire state space into a number of subspaces (the dth dimension is divided into $m_d$ equal subintervals).
Evaluating the points of current population using function $f$
Promising points ← $P_1$ percent of the points of the current population with lowest value of $f$
For each subspace $S_j$ :
Promising rank of $S_j$ ← number of promising points in $S_j$
Promising subspaces ← $P_2$ percent of the most promising subspaces
More precise search of more promising subspaces
More extensive search of more promising subspaces
Evaluating the new points using function $f$
Updating the current population by applying truncation selection on the (current population + newly generated points)
Mutating the $P_3$ percent of current population
If the termination condition reached ?
No
Yes
Returning the best solution found so far

**Figure 3.** Flowchart of the fractal-based algorithm.

```
Algorithm Fractal-based-optimization( $SearchSpace$ , $f$ )
{Inputs:
        $StateSpace$ : the n-dimensional space with boundaries $(L_1, U_1, L_2, U_2, ..., L_n, U_n)$
        $f$ : target function;
 Output:
        $Solution$ : the best solution found by the algorithm;
        ________________________________________________

Initialize the control parameters: $PopSize$, $P_1$, $P_2$, $P_3$, $m_d$ for $1 \le d \le n$
$Pop_0$ = the initial population; // the points generated with uniform distribution over the state space
$i = 0$ ;
$PromisingSubspaces = StateSpace$ ; // assigning whole the search space as the promising area at the beginning
For any dimension $d$ : $1 \le d \le n$
        Partition the dimension $d$ of $StateSpace$ to $m_d$ parts;

while (NOT termination condition)
        {
        $Quality[1..PopSize]$ = the quality of all the points in $Pop_i$ ;// the values of function $f$ for points
        $GoodPoints[1..PopSize]$ = the $P_1$ percent of the points with the highest quality;
        $Count[1..NumberOfSubspaces]$ = the number of good points in each subspace;
        $PromisingRank[1..NumberOfSubspaces] = \frac{Count[1..NumberOfSubspaces]}{PopSize}$
        $PromisingSubspaces$ = the $P_2$ percent of the subspaces with highest promising rank;
        For any promising subspaces $s$
                For any dimension $d$ : $1 \le d \le n$
                        Partition the dimension $d$ of the promising subspace $s$ to $m_d$ parts;

        $Pop_i^{'}[1..PopSize]$ = newly generated points in the subspaces, so that the number of points generated
        in each subspace $s$ is linearly proportional to its promising rank, i.e., $PromisingRank[s]$;
        $Pop_{i+1}[1..PopSize]$ = the best points of $Pop_i$ and $Pop_i^{'}$ selected using truncation selection method;
        Add Gaussian noise to $P_3$ percent of the points randomly selected from $Pop_{i+1}$ ;// mutation
        $i = i + 1$;
        }

$Solution$ =the best solution found so far;

Return $Solution$
        }
```

**Figure 4.** Pseudocode of the fractal-based algorithm.

## 4. EVALUATION AND COMPARISON

To evaluate the algorithm developed in this study, the algorithm is applied to find the optima of a number of continuous benchmark functions. These functions are introduced in Table 1 and depicted in Figure 5. To find the optimum of any of the continuous functions, at first, the function and the ranges of function variables are given to the fractal-based algorithm as inputs (the ranges of function variables indicate the state space boundaries). Then a population of random points is generated within the state space and afterwards, the entire state space is divided into a number of subspaces.

The algorithm continues with iteratively evaluating the points, partitioning the promising subspaces, and generating new points within the subspaces. After evaluating 5000 points, the algorithm terminates and the best point found so far is returned.

The results of the application of the proposed algorithm to the four functions are compared to the results of particle swarm optimization, differential evolution, and genetic algorithms. The parameters of the fractal-based algorithm are set according to Table 2 and the parameters of particle swarm optimization, differential evolution, and genetic algorithms have been configured according to configurations of former studies (Karaboga and Akay, 2009; Civicioglu and Besdok, 2013). They are presented in Table 3. In all the algorithms, the population size is equal to 50 and the termination condition is defined as "reaching the 5000 number of function evaluations".

Each algorithm has been executed 20 times for each benchmark function and a different initial population is used at every run. The algorithms are evaluated based on the two criteria: the mean of the best solutions obtained over 20 runs and the standard deviation of the best solutions obtained over 20 runs. The results are presented in Tables 4 and 5. As mentioned before, these results have been obtained by the algorithms after the same number of function evaluations; thus, the obtained results reflect the speed of algorithms. As it is shown in Table 4, the solutions obtained by fractal-based algorithm are equal to or better than those of other algorithms after doing the same number of function evaluations.

In addition, the convergence diagrams of the fractal-based algorithm for the benchmark functions which obtained by averaging over 20 runs of the algorithm are represents in Figure 6. The distribution of the candidate solutions during the run of the fractal-based algorithm for the four benchmark functions is presented in Figures 7, 8, 9, and 10.

To investigate the effect of parameter $m_i$ (the number of intervals for dividing a subspace dimensions) on the algorithm performance, the algorithm runs for several values of $m_i$ ($i = 1, 2$). The results averaged over 50 runs for the four benchmark functions are presented in Table 6. For very large values of $m_i$, in every iteration of algorithm the promising subspaces are divided into a large number of subspaces; thus, the algorithm extremely focuses on the small subspaces that have a large number of promising points. Therefore, it is not able to explore the entire state space sufficiently to find the optimum. On the other hand, for very small values of $m_i$, the algorithm is not able to focus on the fine parts of state space; thus it may merely be able to reach the neighbourhood around the optimum and only return the best point among the points generated in that neighbourhood. Therefore, it is probable that the algorithm does not find the global optimum before reaching the termination condition or convergence. As it is shown in Table 6, the mean of the best solutions obtained over 20 runs are far from the functions global optima, for very large and very small values of $m_i$.

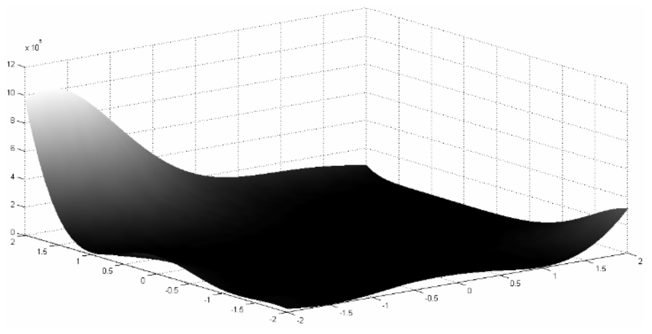
To study the effect of parameter $P_3$ (the rate of random modification of solutions) on the convergence of the fractal-based algorithm, several values of $P_3$ have been examined. The number of iterations required for algorithm convergence averaged over 20 runs for the four benchmark functions are presented in Table 7. As it is concluded from these results, assigning low values to parameter $P_3$ accelerates the algorithm convergence. On the other hand, the high value of parameter $P_3$ leads to finding better solutions because it prolongs the exploration phase of algorithm before the algorithm convergence.

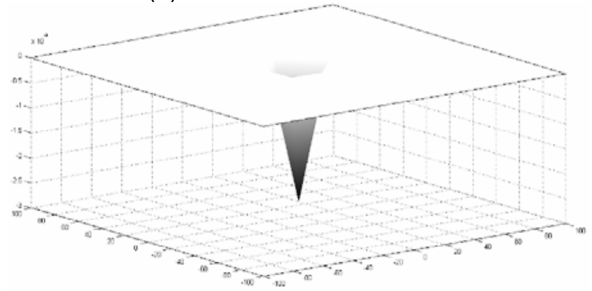
*Table 1:* Benchmark functions used to evaluate the proposed algorithm and their specifications (Karaboga and Akay, 2009; Maeda and Tsuda, 2015).

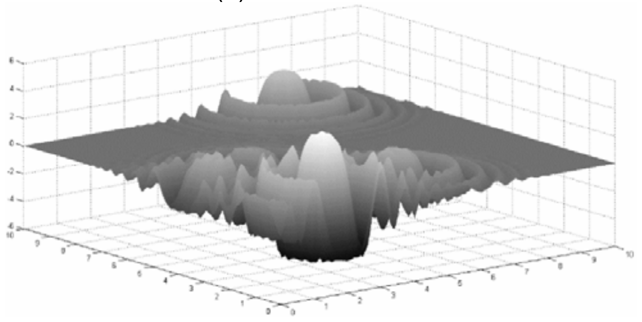
| Function name | Function definition | Range | Minimum value |
|---|---|---|---|
| Goldstein -Price | $f(x_1,x_2)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)].$ $[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$ | [-2, 2] | 3 |
| Easom | $f(x_1,x_2)=-\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2)$ | [-100, 100] | -1 |
| Langermann | $f(x)=\sum_{i=1}^{m}c_i\exp\left[-\frac{1}{\pi}\sum_{j=1}^{n}(x_j-a_{ij})^2\right]\cos[\pi\sum_{j=1}^{n}(x_j-a_{ij})^2]$ $a=[3,5,2,1,7],\quad b=[5,2,1,4,9],\quad c=[1,2,5,2,3]$ | [0,10] | $-1.08093$ |
| Shubert | $f(x_1,x_2)=-\sum_{i=1}^{5}i\cos((i+1)x_1+1)\sum_{i=2}^{5}i\cos((i+1)x_2+1)$ | [-10,10] | $-186.73090$ |

*Table 2:* Parameters values of fractal-based algorithm.

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Population size | 50 | -------- |
| $P_1$ | 60 % | The percentage of solutions which are selected as promising points |
| $P_2$ | 30 % | The percentage of subspaces which are selected as promising subspaces |
| $P_3$ | 5 % | The percentage of solutions which are selected randomly to be modified |
| $m_1$ , $m_2$ | 10 | The number of intervals for dividing the first and second dimensions of a subspace |
| Maximum number of function evaluations | 5000 | -------- |


(a) Goldstein-Price function


(b) Easom function


(c) Langermann  function

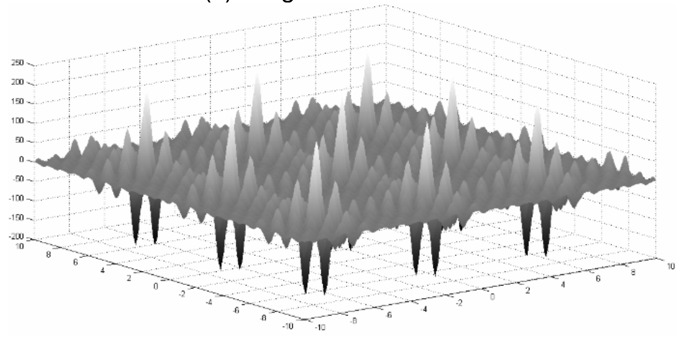

(d) Shubert function

**Figure 5.** Graphical representation of four benchmark functions.

*Table 3:* Parameters valuse of particle swarm optimization, differential evolution, and genetic algorithms, being inspired by former studies (Karaboga and Akay, 2009; Civicioglu and Besdok, 2013).

| **Algorithm** | **Parameters** | **Values** |
|---|---|---|
| **Particle swarm optimization** | Cognitive component | 1.8 |
| | Social component | 1.8 |
| | Inertia weight | 0.6 |
| | Population size | 50 |
| | Termination condition | Reaching the 5000 number of function evaluations |
| **Differential evolution algorithm** | Differential weight | 0.5 |
| | Crossover rate | 0.9 |
| | Population size | 50 |
| | Termination condition | Reaching the 5000 number of function evaluations |
| **Genetic algorithm** | Crossover method | Single point crossover |
| | Crossover rate | 0.8 |
| | Mutation rate | 0.01 |
| | Selection method | Stochastic uniform sampling |
| | Replacement rate | 0.1 |
| | Population size | 50 |
| | Termination condition | Reaching the 5000 number of function evaluations |

*Table 4:* The mean of the best solutions obtained over 20 runs of the fractal-based algorithm and the four algorithms compared.

| | **Particle swarm optimization** | **Differential evolution algorithm** | **Genetic algorithm** | **Fractal-based algorithm** |
|---|---|---|---|---|
| **Goldstein-Price** | 2.988 | 2.996 | 2.965 | 2.996 |
| **Easom** | -1 | -1 | -1 | -1 |
| **Langermann** | −1.08084 | −1.08091 | −1.08087 | −1.08091 |
| **Shubert** | -186.7283 | -186.7295 | -186.7288 | −186.7297 |

*Table 5:* The standard deviation of best solutions obtained over 20 runs of the fractal-based algorithm and the four algorithms compared.

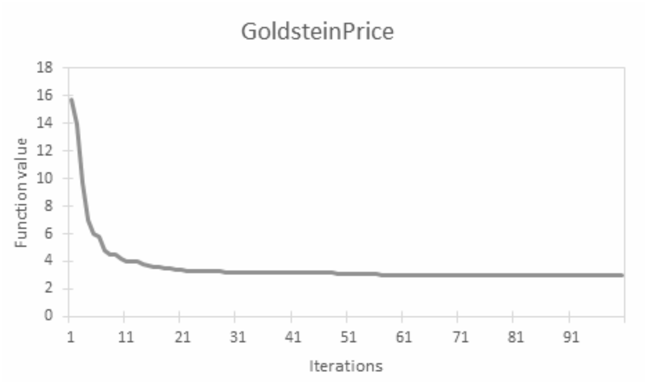
| | **Particle swarm optimization** | **Differential evolution algorithm** | **Genetic algorithm** | **Fractal-based algorithm** |
|---|---|---|---|---|
| **Goldstein-Price** | 0.017 | 0.007 | 0.052 | 0.005 |
| **Easom** | 0 | 0 | 0 | 0 |
| **Langermann** | 0.00021 | 0.00003 | 0.00015 | 0.00004 |
| **Shubert** | 0.0013 | 0.0003 | 0.0011 | 0.0004 |

*Table 6:* The mean of the best solutions obtained over 20 runs of the fractal-based algorithm for several values of parameters $m_1$ and $m_2$. Other parameters were adjusted according to Tables 2 and 3.

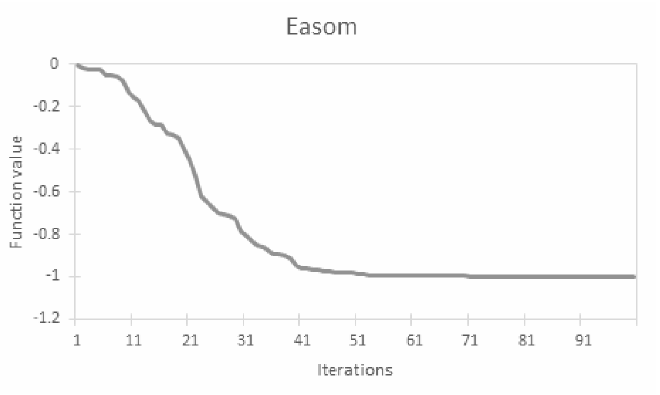
| | $m_1, m_2 = 2$ | $m_1, m_2 = 5$ | $m_1, m_2 = 10$ | $m_1, m_2 = 15$ | $m_1, m_2 = 20$ |
|---|---|---|---|---|---|
| **Goldstein-Price** | 2.981 | 2.996 | 2.996 | 2.992 | 2.984 |
| **Easom** | -0.989 | -1 | -1 | -1 | -1 |
| **Langermann** | −1.08086 | −1.08092 | −1.08091 | −1.08091 | −1.08087 |
| **Shubert** | −186.7283 | −186.7294 | −186.7297 | −186.7297 | −186.7295 |

*Table 7:* The mean of the best solutions obtained over 20 runs of the fractal-based algorithm for several values of parameter $P_3$. Other parameters were adjusted according to Tables 2 and 3.

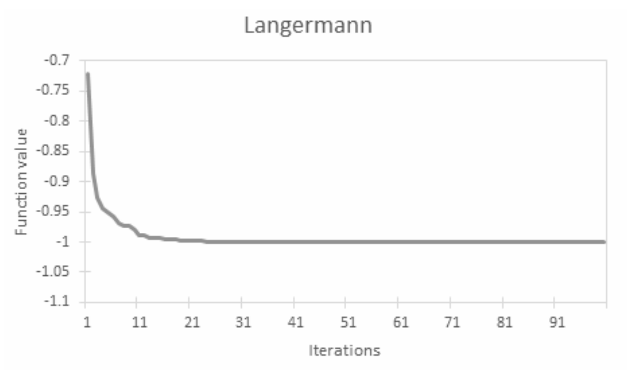
| | $P_3 = 2$ | | $P_3 = 3$ | | $P_3 = 5$ | | $P_3 = 7$ | |
|---|---|---|---|---|---|---|---|---|
| | Best solution | Number of iterations | Best solution | Number of iterations | Best solution | Number of iterations | Best solution | Number of iterations |
| **Goldstein-Price** | 2.986 | 49.8 | 2.988 | 63.1 | 2.996 | 65.3 | 2.996 | 67.5 |
| **Easom** | -0.993 | 65.4 | −1 | 68.5 | −1 | 75.7 | −1 | 77.4 |
| **Langermann** | −1.08083 | 31.6 | −1.08085 | 35.2 | −1.08091 | 37.3 | −1.08092 | 39.6 |
| **Shubert** | −186.7293 | 63.4 | −186.7295 | 64.6 | −186.7297 | 67.9 | −186.7297 | 68.1 |



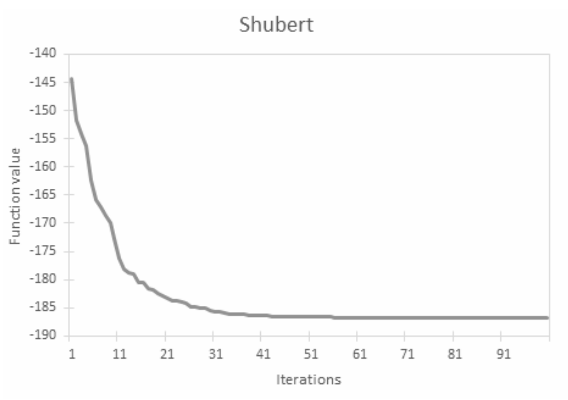
(a) Convergence diagram for Goldstein-Price function



(b) Convergence diagram for Easom function



(c) Convergence diagram for Langermann function



(d) Convergence diagram for Shubert function

**Figure 6.** Algorithm convergence diagrams for the four benchmark functions.

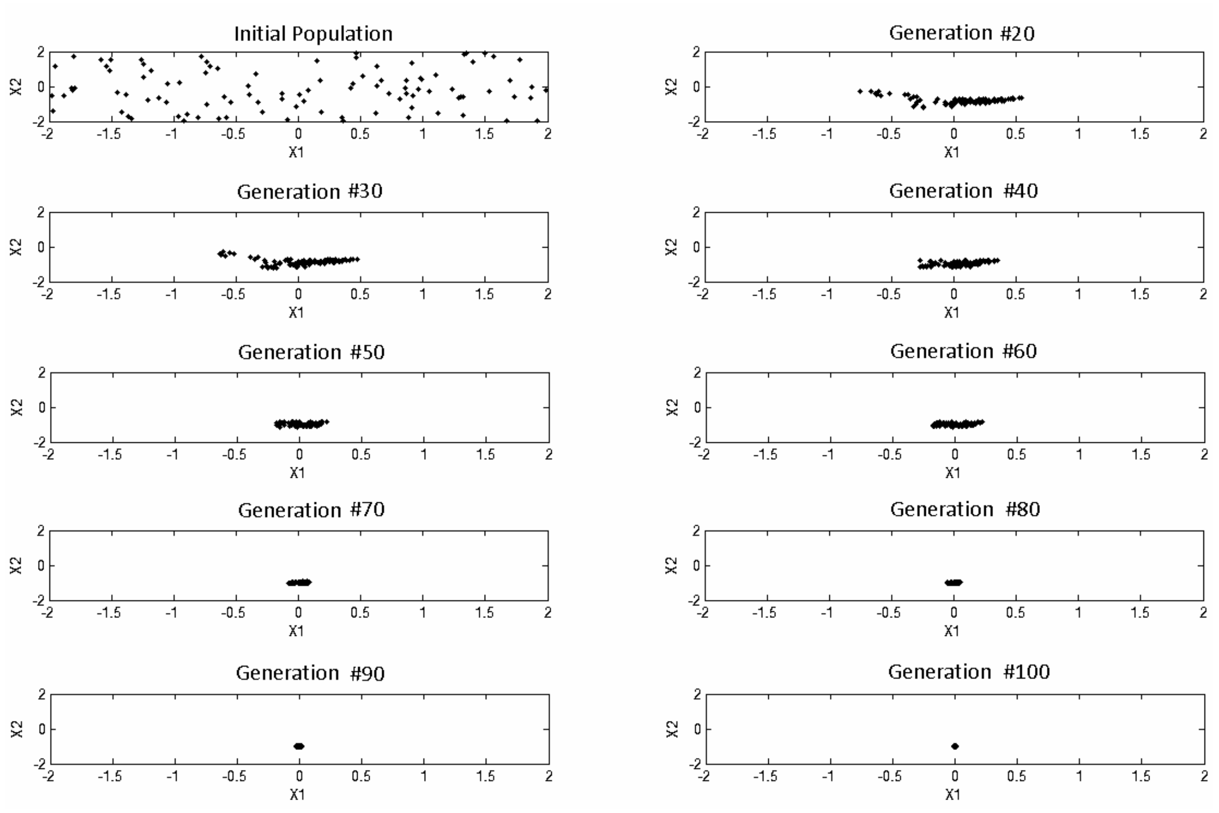


**Figure 7.** Distribution of candidate solutions during the run of the fractal-based algorithm for the Goldstein-Price function.

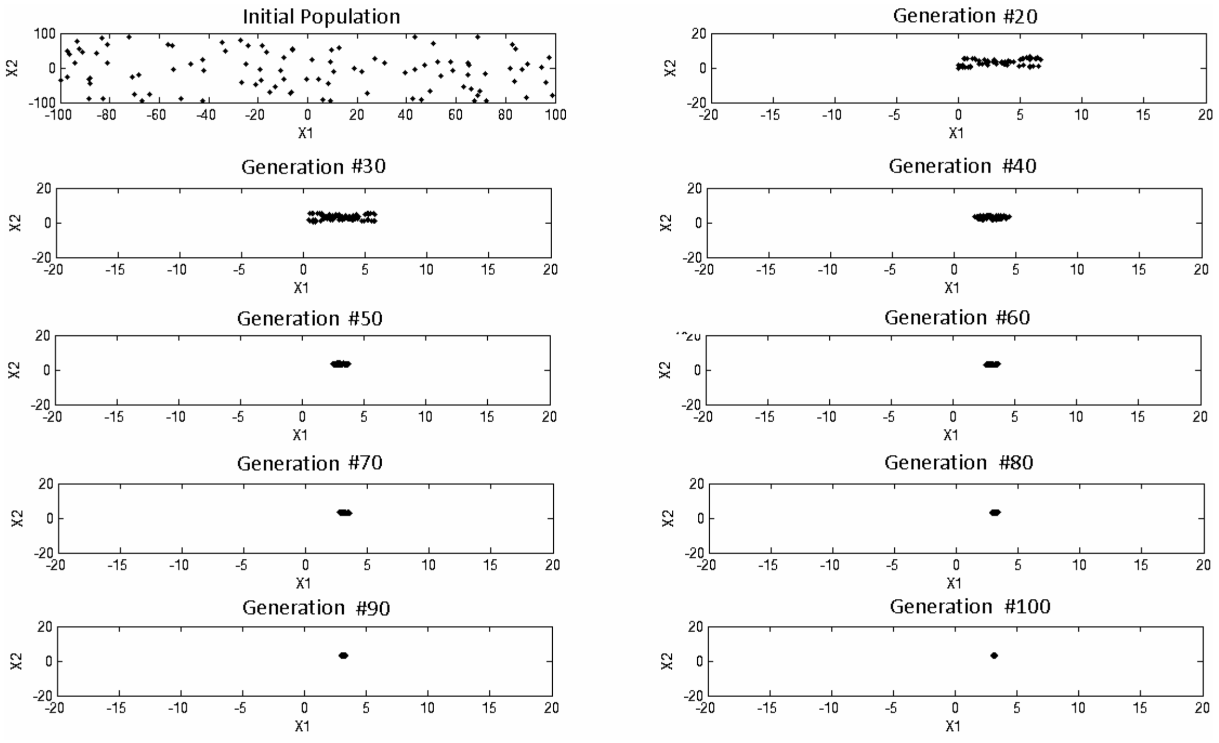


**Figure 8.** Distribution of candidate solutions during the run of the fractal-based algorithm for the Easom function.

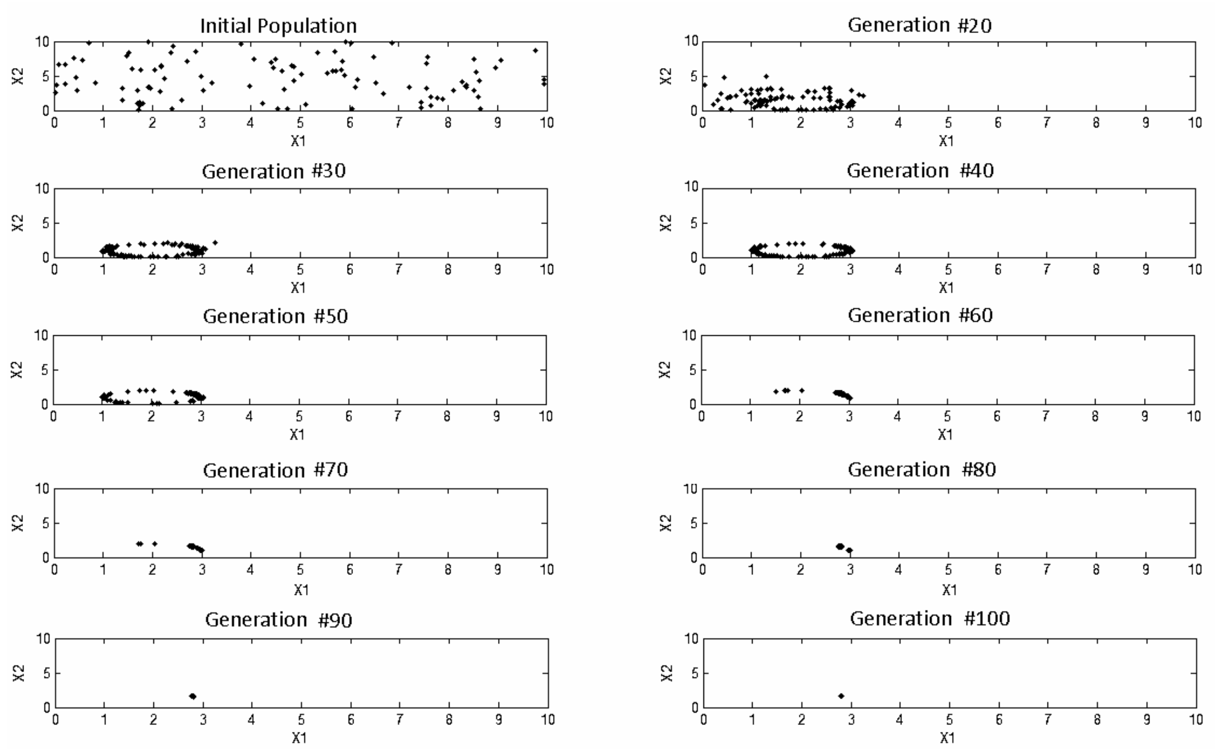


**Figure 9.** Distribution of candidate solutions during the run of the fractal-based algorithm for the Langermann function.

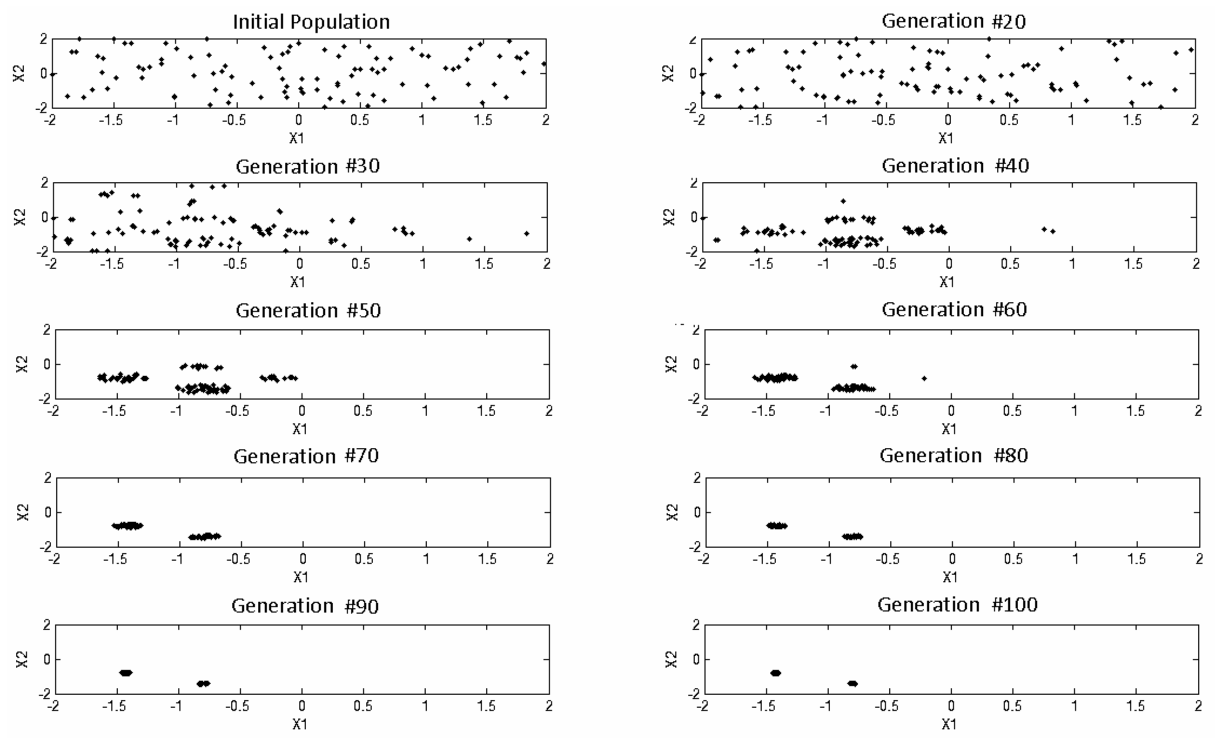


**Figure 10.** Distribution of candidate solutions during the run of the fractal-based algorithm for the Shubert function.

## 5. CONCLUSION

This paper developed a population-based metaheuristic algorithm for the continuous optimization problem, where, by iterative partitioning of the state space based on fractal-shaped structures, the promising areas of the state space are estimated so that those areas are more precisely and more extensively searched. The proposed algorithm was compared to particle swarm optimization, differential evolution, and genetic algorithms and the results of the evaluations demonstrated that the proposed algorithm can find the solution to the benchmark problems with higher precision within appropriate time. The main advantage of the fractal-based algorithm is that during the search process it hierarchically partitions the state space into small blocks and it pays less attention to the less promising blocks and focuses on the promising blocks. It this way, the algorithm conducts a fast and targeted search towards the problem solution.

It is recommended that future studies further improve the heuristic proposed in this paper to detect the promising areas. For this purpose, it is suggested that besides the number of good points in an area, the future studies pay attention to criteria such as the mean and standard deviation of the high quality points existing in each area so that the promising areas are estimated more precisely. In addition, in the future studies the proposed algorithm can be extended to be applied in dynamic optimization problems and its performance can be compared to former population-based algorithms proposed for dynamic optimization (Kaedi, et al., 2013; Kaedi, et al., 2016). Furthermore, more research should be conducted on choosing suitable values for the control parameters of the proposed algorithm.

## REFERENCES

Ali, M. Z., Awad, N. H., Duwairi, R. M., 2016, Multi-objective differential evolution algorithm with a new improved mutation strategy, *International Journal of Artificial Intelligence,* **14(2)**, 23-41.

Ayan, K., Kılıç, U., Baraklı, B., 2015, Chaotic artificial bee colony algorithm based solution of security and transient stability constrained optimal power flow, *International Journal of Electrical Power & Energy Systems*, **64**, 136–147.

Azar, D., Fayad, K., Daoud, C., 2016, A combined ant colony optimization and simulated annealing algorithm to assess stability and fault-proneness of classes based on internal software quality attributes, *International Journal of Artificial Intelligence,* **14(2)**, 137-156.

Bäck, T., Fogel, D., Michalewicz, Z., 1997, *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, Bristol, UK.

Castillo, O., Neyoy, H., Soria, J., Melin, P., Valdez, F., 2015, A new approach for dynamic fuzzy logic parameter tuning in ant colony optimization and its application in fuzzy control of a mobile robot, *Applied Soft Computing*, **28**, 150–159.

Chávez-Conde, E., Valdez, S. I., Hernández, E., 2015, Concurrent Structure-Control Design of Parallel Robots Using an Estimation of Distribution Algorithm, Multibody Mechatronic Systems, *The Series of Mechanisms and Machine Science*, **25**, 315-325.

Chiong, R., 2009, Nature-Inspired Algorithms for Optimisation, *Studies in Computational Intelligence*, Springer, Heidelberg, **193**, 1-50.

Civicioglu, P., Besdok, E., 2013, A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms, *Artificial Intelligence Review*, **39(4)**, 315-346.

Dorigo, M., 1992, Optimization, Learning and Natural Algorithms, PhD thesis, DEI, Politecnico di Milano, Italy.

Ghosn, S. B., Drouby, F., Harmanani, H. M., 2016, A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves, *International Journal of Artificial Intelligence,* **14(1)**, 130-144.

Glotić, A., Zamuda, A., 2015, Short-term combined economic and emission hydrothermal optimization by surrogate differential evolution, *Applied Energy*, **141**, 42-56.

Glover, F., 1999, *Scatter Search and Path Relinking*, in: D. Corne, M. Dorigo, F.Glover, (eds.), New Methods in Optimization, McGraw-Hill, Maidenhead, UK, England, 291-316.

Glover, F., Laguna, M., Martí, R., 2000, Fundamentals of Scatter Search and Path Relinking, *Control and Cybernetics*, 29**(3)**, 653-684.

Gotmare, A., Patidar, R., George, N. V., 2015, Nonlinear system identification using a cuckoo search optimized adaptive Hammerstein model, *Expert Systems with Applications*, 42**(5)**, 2538-2546.

Holland, J., 1975, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

Jung, H. C., Kim, J. S., Heo, H., 2015, Prediction of building energy consumption using an improved real coded genetic algorithm based least squares support vector machine approach, *Energy and Buildings*, **90**, 76-84.

Kaedi, M., Ghasem-Aghaee, N., Ahn C. W., 2016, Biasing the transition of Bayesian optimization algorithm between Markov chain states in dynamic environments, *Information Sciences*, **334-335**, 44-64.

Kaedi, M., Ghasem-Aghaee, N., Ahn C. W., 2013, Holographic memory-based Bayesian optimization algorithm (HM-BOA) in dynamic environments, *Science China Information Sciences*, **56 (9)**, 1-17.

Karaboga, D., 2005, *An idea based on honey bee swarm for numerical optimization*, Technical Report, tr06, Engineering Faculty, Computer Engineering Department, Erciyes University, Kayseri, Turkey.

Karaboga, D., Akay, B., 2009, A comparative study of artificial bee colony algorithm, *Applied Mathematics and Computation*, 214**(1)**, 108-132.

Kennedy, J., Eberhart, R., 1995, Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, 1942-1948.

Kennedy, J., Eberhart, R., 2001, *Swarm Intelligence*, Morgan Kaufmann Publisher Inc., San Francisco, USA.

Larranaga, P., Lozano, J. A., 2002, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Boston, MA.

Maeda, M., Tsuda, S., 2015, Reduction of artificial bee colony algorithm for global optimization, *Neurocomputing*, **148**, 70–74.

Manikandan, S., Ramar, K., Iruthayarajan, M. W., Srinivasagan, K. G., 2014, Multilevel thresholding for segmentation of medical brain images using real coded genetic algorithm, *Measurement*, **47**, 558-568.

Martí, R., Corberán, A., Peiró, J., 2015, Scatter search for an uncapacitated p-hub median problem, *Computers & Operations Research*, **58**, 53-66.

Martin, D., Caballero, B., Haber, R., 2009, Optimal tuning of a networked linear controller using a multi-objective genetic algorithm and its application to one complex electromechanical process, *International Journal of Innovative Computing, Information and Control*, **5(10)**, 3405-3414.

Moraes, A. O. S., Mitre, J. F., Lage, P. L. C., Secchi, A. R., 2015, A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems, *Applied Mathematical Modellin*g, **39(14)**, 4223-4241

Nakrani, S. Tovey, C., 2004, On honey bees and dynamic server allocation in Internet hosting centers, *Adaptive Behavior*, **12(3-4)**, 223-240.

Nicoară, E. S., 2012, Population-based metaheuristics: A comparative analysis, *International Journal of Science and Engineering Investigations*, **1(8)**, 84-88.

Pelikan, M., Sastry, K., Cantu-Paz, E., 2006, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, *The series of Studies in Computational Intelligence*, Springer-Verlag, Berlin, Heidelberg.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim S., Zaidi, M., 2005, *The bees algorithm, Technical Note*, Manufacturing Engineering Center, Cardiff University, UK.

Precup, R.-E., David, R.-C., Petriu, E. M., Preitl, S., Radac, M.-B., 2014, Novel adaptive charged system search algorithm for optimal tuning of fuzzy controllers, *Expert Systems with Applications*, **41(4)**, 1168-1175.

Qi, H., Niu, C. Y., Gong, S., Ren, Y. T., Ruan, L. M., 2015, Application of the hybrid particle swarm optimization algorithms for simultaneous estimation of multi-parameters in a transient conduction-radiation problem, *International Journal of Heat and Mass Transfer*, **83**, 428-440.

Raja, S. B., Srinivas Pramod, C. V., Krishna, K. V., Ragunathan, A., Vinesh, S., 2015, Optimization of electrical discharge machining parameters on hardened die steel using Firefly Algorithm, *Engineering with Computers*, **31(1)**, 1-9.

Ramírez-Ortegón M. A., Märgner V., Cuevas E., Rojas R., 2013, An optimization for binarization methods by removing binary artifacts, *Pattern Recognition Letters*, **34(11)**, 1299-1306.

Spall, J. C., 2003, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, John Wiley and Sons, Hoboken, NJ.

Storn, R., Price, K., 1997, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, **11(4)**, 341-359.

Talbi, E., 2009, *Metaheuristics from Design to Implementation*, John Wiley and Sons, Hoboken, NJ.

Wang, Sh., Wang, L., Liu, M., Xu, Y., 2015, An order-based estimation of distribution algorithm for stochastic hybrid flow-shop scheduling problem, *International Journal of Computer Integrated Manufacturing*, **28(3)**, 307-320.

Yang, X. S., 2010, *Nature Inspired Metaheuristic Algorithms*, Second Edition, Luniver Press, UK.

Yang, X. S., Deb, S., 2009, Cuckoo search via Lévy flights, *World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)*, Coimbatore, India, 210-214.

Zäpfel, G., Braune, R., 2010, *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*, Springer-Verlag, Berlin, Heidelberg.