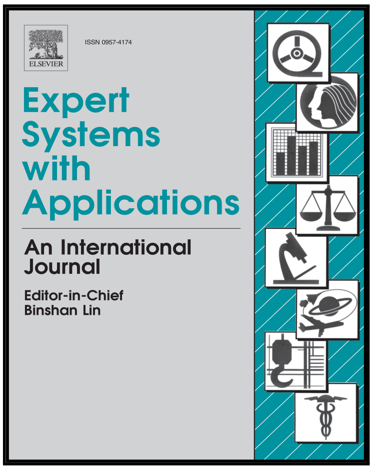
Accepted Manuscript



Bernstain-Search Differential Evolution Algorithm for Numerical Function Optimization

Pinar Civicioglu, Erkan Besdok

PII: S0957-4174(19)30533-0
DOI: https://doi.org/10.1016/j.eswa.2019.112831
Article Number: 112831
Reference: ESWA 112831

To appear in: *Expert Systems With Applications*

Received date: 20 January 2019
Revised date: 19 July 2019
Accepted date: 20 July 2019

Please cite this article as: Pinar Civicioglu, Erkan Besdok, Bernstain-Search Differential Evolution Algorithm for Numerical Function Optimization, *Expert Systems With Applications* (2019), doi: https://doi.org/10.1016/j.eswa.2019.112831

Highlights

- Bernstein polynomials based parameter-free crossover
- A new universal / parameter-free Differential Evolution
- Real-valued numerical function optimization
- Evolutionary Image Vectorization
- Evolutionary Digital Terrain Model Simplification

Available online at www.sciencedirect.com

ScienceDirect

Expert Systems with Applications 00 (2019) 1–??

Bernstein-Search Differential Evolution Algorithm for Numerical Function Optimization

Pinar Civicioglu[1]

Erciyes University, Faculty of Aeronautics and Astronautics, Dept. of Aircraft Electrics and Electronics, Kayseri, Turkey

Erkan Besdok[2,*]

Erciyes University, Faculty of Engineering, Dept. of Geomatics Eng., Kayseri, Turkey

Abstract

The standard Differential Evolution Algorithm (sDE) is a stochactic search method commonly used in evolutionary computing. The problem solving success of sDE is highly sensitive to the *genetic operators* used and the initial values of the parameters of these operators. Since a universal Differential Evolution Algorithm (uDE) is not sensitive to the structure and parameter values of the *genetic operators* used, it is parameter-free in practice and easier to control than sDE. uDE does not need a *trial-and-error* process when selecting the *genetic operators* and initial values of intrinsic parameter of related *genetic operators* to solve the problem, unlike the sDE. Therefore, the use and adaptation of a uDE to solve different types of numerical engineering problems is easy and time-consuming compared to sDE. In this paper, a new uDE, *Bernstein-Search Differential Evolution Algorithm* (BSD), is introduced. BSD is new and easily controllable, simple structured, non-recursive, highly efficient, fast and practically parameter-free uDE. BSD have a too feasible random *crossover* and *mutation* process and does not have a control-parameter setting process, contrary to sDE and its improved variants. In this paper, 30 benchmark problems of CEC'2014, 60 classic benchmark problems, image evolution problems for 12 test images and one Triangulated Irregular Network (TIN) refinement problem were used in the experiments performed to investigate the problem solving success of BSD, statistically. Four tested methods (*i.e.*, ABC, JADE, CUCKOO, WDE) were used in the conducted experiments. Problem solving successes of BSD and tested methods were statistically compared by using Wilcoxon Signed Rank Test piecewisely. Results obtained from the performed tests showed that in general, problem solving success of BSD is *statistically better* than the tested methods that have been used in this paper.

Keywords: Artificial Bee Colony Algorithm, Differential Evolution Algorithm, Cuckoo Search Algorithm, Weighted Differential Evolution Algorithm, Image Evolution.

*Corresponding author

Email addresses: civici@erciyes.edu.tr (Pinar Civicioglu), ebesdok@erciyes.edu.tr (Erkan Besdok)

[1]Tel.: +90 352 207 66 66-41054; fax: +90 352 437 57 44.

[2]Tel.: +90 352 207 66 66-32650; fax: +90 352 437 57 84.

1. Introduction

Standard Differential Evolution Algorithm (sDE) is one of the most widely used stochastic search method in evolutionary computing. The *genetic operators* (*i.e.*, *mutation*, *crossover*) used and the initial values of the intrinsic parameters of these operators affect the problem solving success of sDE (Brest et al , 2006; Das, Mullick, & Suganthan , 2016; Mohamed, & Suganthan , 2018; Price, Storn, & Lampinen , 2005). Selecting genetic operators in sDE and adjusting initial values of their parameters is based on a *trial-and-error* process and it is time-consuming (Zhang, & Sanderson , 2009; Mlakar, & et al , 2016). The chaotic relationship between the initial values of the parameters of evolutionary algorithms and the problem-solving successes of the relevant algorithms is still an active research area since there is no analytical parameter setting method that sets the best Evolutionary Search Algorithm (EA) parameters for the problem to be solved. Since EAs are too sensitive to initial values of intrinsic parameters, using fixed initial values for related intrinsic parameters may limit the problem solving success of EAs. Therefore, various adaptive parameter adjustment methods have been developed for EAs. Adaptive parameter setting methods are also effective on problem solving success of EA depending on the problem type, such as fixed parameter setting methods (Karaboğa, & Basturk B , 2007; Lynn, & Suganthan , 2017; Qin et al , 2014; Chen, & et al , 2017; Zhang, & Zhu , 2011; Liu & et al. , 2018; Özsoydan, Baykaşoğlu , 2019; Clerc, & Kennedy , 2002).

There are many *mutation* and *crossover* operators developed for sDE. *Mutation* operators generally produce a new solution-vectors by using one of the two different methods. The first type of solution-vector generation methods are simply based on the parameter-space based vector blending (Das, Mullick, & Suganthan , 2016; Opara, & Arabas , 2018). Although these methods allow to produce a new vector that is very different from parent vectors, its success depends on *gene diversity* in parent vectors (Das, Mullick, & Suganthan , 2016; Opara, & Arabas , 2018; Zhang et al , 2019). Methods of generating the second type of solution-vector are based on producing the new solution-vector by using a pseudo random number generator. These methods are not affected by the problem of *gene-diversity*, but the search process is prolonged and often becomes inefficient (Das, Mullick, & Suganthan , 2016; Zhang et al , 2019). There are two types of *crossover* operators called *binomial* and *exponential* for sDE (Das, Mullick, & Suganthan , 2016). Unfortunately, there is no analytical method to show which *mutation* and *crossover* operators should be used to solve a numerical problem most efficiently by using sDE. Therefore, it is difficult and time consuming to determine the combination of *mutation* and *crossover* operators required to operate the relevant search process in the most efficient manner. Developing more efficient *mutation* and *crossover* processes can make the relevant search process radically efficient (Das, Mullick, & Suganthan , 2016; Zhang et al , 2019).

The parameters of EAs can be divided into two groups; *common parameters* and *structural parameters*. *Common parameters* are the parameters that cannot be structurally related to the direct algorithm, such as the number of iterations, the number of elements of the *pattern matrix* (i.e., *population* in raw-genetic methods), and the size of problem. The values of these parameters all affect the success of problem solving in all EAs. *Structural parameters*, which are directly necessary to define the relevant algorithm, affect the problem solving success of the algorithm and its convergence ability. Apart from common parameters, EAs have other parameters that make their use difficult and require time-consuming parameter adjustment processes (Civicioglu , 2013a, 2012, 2013b; Price, Storn, & Lampinen , 2005; Karaboğa, & Basturk B , 2007).

The parameter adaptation methods developed for EAs can be divided into 3 groups; *deterministic methods*, *adaptive methods* and *self-adaptive methods*. *Deterministic methods* change the parameter by using analytical methods without using the information provided by the algorithm. *Deterministic methods* do not have the ability to change behavior depending on the type of problem. Cuckoo Search Algorithm (CUCKOO) is a very successful and very fast differential evolution algorithm (Civicioglu, & Besdok , 2013, 2014; Yang, & Deb , 2009). The parameters of CUCKOO have fixed values. Therefore, it uses a deterministic process for tuning the parameters. *Adaptive methods* adjust the values of the corresponding control parameters according to the success of the algorithm. Adaptive Differential Evolution Algorithm (JADE) (Zhang, & Sanderson , 2009) and SADE (Brest et al , 2006) are DE algorithms with the ability to set adaptive parameters. *Adaptive methods* are capable of changing behavior depending on the success achieved. *Self-adaptive methods* determine the value of a parameter separately for each *pattern vector*. Artificial Bee Colony Algorithm (ABC) is such an algorithm (Karaboğa, & Basturk B , 2007; Civicioglu , 2013a,b). In the adaptive parameter determination method, the *pattern vectors* with good parameter values have the potential to better evolve. Using an adaptive or self-adaptive method to determine the relevant parameter values of an EA may improve the problem-solving success of the respective EA. This cannot completely eliminate the *trial-and-error* process required to determine the parameter values, but it can be quite shortened. Adaptive and Self-Adaptive DEs are more successful than classic DE in solving numerical optimization problems. Even if parameter values are adaptively determined, finding the best *mutation* and *crossover* strategy that should be used to solve the problem involves a time-consuming *trial-and-error* process (Civicioglu , 2013a, 2012, 2013b). This motivates the development of new DE techniques whose parameters can be randomly determined, and the *mutation* and *crossover* processes are relatively simple.

A universal DE (uDE) that is free from deficiencies caused by the genetic operators of sDE can search the search space more efficiently. The problem solving success of a uDE is not sensitive to the genetic operators used and the initial values of the parameters of the relevant operators. uDE development efforts aim to develop highly advanced DE versions that are parameter-free, very efficient, fast and easier to control. The need for an easily controllable, simple structured, non-recursive, highly efficient, fast and practical parameter-free uDE has motivated efforts to develop the proposed algorithm, BSD.

In this paper, ABC, JADE, CUCKOO and WDE were used as tested methods due to their good problem solving abilities.

ABC has been used in many studies and its problem-solving success is better than many of the EAs (Karaboğa, & Basturk B , 2007; Civicioglu , 2013a, 2012, 2013b). ABC is a structurally adaptive algorithm that tends to investigate more efficient areas of search space in more detail. Inefficient search space segments are abandoned after a certain number of searches. During this time, using unboundend search may require minor structural changes. ABC searches separately for each *pattern vector* parameter. Therefore, the *pattern vectors* in the next generation *pattern matrix* cannot be determined naturally before the search process for a *pattern vector* is completed.

JADE is an extremely successful and highly developed adaptive DE (Zhang, & Sanderson , 2009). JADE can converge to the problem solution at an astonishing speed (Zhang, & Sanderson , 2009). The structure of JADE is quite complex. JADE can do bounded / unbounded search. In JADE, unbounded search does not require a structural modification.

CUCKOO has a two-step search process (Civicioglu, & Besdok , 2013, 2014; Yang, & Deb , 2009). CUCKOO has the ability to converge surprisingly fast to the result of the problem. CUCKOO uses *levy-fly* rules. Therefore, the evolutionary step can change the magnitude value (*i.e.*, the scale value) and the direction of evolution, rapidly. This makes it very easy for CUCKOO to escape from local solutions. CUCKOO can do bounded / unbounded search.

A random determination of EA's parameter values means that it does not have a parameter in practice. The values of the intrinsic parameters of Weighted Differential Evolution Algorithm (WDE) (Civicioglu, Besdok, & et al , 2018) are randomly determined. Therefore, WDE is a structurally parameter-free method, in practice. WDE is more successful in solving various numerical problems than JADE (Zhang, & Sanderson , 2009), ABC (Karaboğa, & Basturk B , 2007), BSA (Civicioglu , 2013a), and CUCKOO (Civicioglu, Besdok, & et al , 2018).

The structural parameter values of the BSD introduced in this paper are determined randomly. BSD is a very simple structured row-genetic DE. In BSD, each *pattern vector* in the *pattern matrix* is evolved separately. Since the evolution of each *pattern vector* is independent from the other, BSD is naturally a parallel search algorithm. In BSD, the *crossover* process is controlled using Bernstain polynomials. Therefore BSD does not have a parameter for the *crossover* process. The problem-solving success of BSD, like other EAs, is sensitive to common parameter values.

Rapidly developing artificial intelligence (AI) technologies, (*i.e.*,*Expert systems*, *Statistical Learning*, *Deep Learning and Artificial Neural Networks*, *Fuzzy Systems*, *Evolutionary Computation*) have the potential to transform the entire social and economic structure of human society into a new form in the near future. Many advanced AI applications still require new and highly advanced portable / wearable nano-scaled computers / sensors, energy technologies and computing algorithms. Therefore, even the smallest contributions of researchers to evolutionary computing technologies are critical to technological development. In recent years, researchers have focused on developing rapid EA methods with more efficient genetic operators.

The innovations provided by BSD are as follows;

- BSD uses a unique bijective *mutation* strategy.
- BSD uses a more efficient *crossover* operator than those of sDE's *crossover* operators.
- The *crossover* process of BSD is controlled randomly by using Bernstein polynomials.
- BSD does not have a pre-fixed *mutation* and *crossover* rate value, in contrary to sDE.
- Since BSD is a non-recursive method, it is easy using BSD with *parallel-computing* methods.
- BSD does not require parameter tuning phases for intrinsic parameters of *mutation* and *crossover* operators, in contrary to sDE and its modern variants.
- BSD is a partially-elitist method, in contrary to elitist methods (i.e.,ABC, JADE, and CUCKOO).

This paper is organized as follows: In Section 2, Bernstain-Search Differential Evolution Algorithm (BSD) is expressed. In Section 3, Experiments are given and finally in Section 4, Conclusions are presented.

2. Bernstain-Search Differential Evolution Algorithm (BSD)

In the literature of evolutionary algorithms, a random solution is called a *pattern vector* and N *pattern vectors* form the *pattern matrix* P. Each *pattern vector* consists of D *individuals*. EAs can perform *bounded* and/or *unbounded* search. Bounded search works between the upper and lower limits of the individuals (Civicioglu , 2013a, 2012, 2013b; Civicioglu, Besdok, & et al , 2018). BSD is designed as a global minimizer algorithm that performs bounded search.

In BSD, individuals are determined using Eq 1;

$$P_{i,j} \sim \mathbf{U}(low_j, up_j) \mid i = [1:N], j = [1:D], \;\; i,j \in \mathbb{Z}^+ \tag{1}$$

The objective function values of the *pattern vectors* are calculated using Eq 2;

$$fitP_i = \mathcal{F}(P_i) \tag{2}$$

The global minimizer *pattern vector*, *bestP*, which provides the best solution to the problem, and the objective function value of the global minimizer *pattern vector*, *solP*, are obtained with Eq 3;

$$[solP, bestP] = [fitP_{(\gamma)}, P_{(\gamma)}] \;\mid\; fitP_{(\gamma)} = \min(fitP) \;\mid\; \gamma \in [1:N] \tag{3}$$

BSD controls the *crossover* ratio with M by using Eq 4-Eq 5. The initial value of M is determined by using Eq 4.

$$M_{(i=1:N, j=1:D)} = 0 \tag{4}$$

$$M_{(i,u(1:\lceil \rho \cdot D \rceil))} = 1 \tag{5}$$

Here, ρ is defined using Eq 6;

$$\begin{array}{lll} switch & \kappa_0 & \\ & case\ 1 & \rho = (1-\beta)^2 \\ & case\ 2 & \rho = 2 \cdot \beta \cdot (1-\beta) \\ & case\ 3 & \rho = \beta^2 \\ endsw & & \end{array} \tag{6}$$

where $\beta \sim \mathbf{U}(0,1)$ and $\kappa_0 = \lceil 3 \cdot \kappa_1^3 \rceil, \kappa_1 \sim \mathbf{U}[0\ 1], \quad \kappa_0 \in \mathbf{U}\{1:3\}$. In the Eq 6, the ρ value is computed by using 2^{nd} degree Bernstain polynomials (Azhari & et al , 2018). The 2^{nd} degree Bernstain polynomials are described in Subsection 2.2, Bernstain Polynomials.

The u vector, in Eq 5, is defined by using Eq 7;

$$\mathrm{u} = permute(1:\mathrm{D}) \tag{7}$$

Here, the $permute(\cdot)$ function randomly changes the order of the elements of $(\cdot)$. The *evolutionary step size*, F, is determined by using Eq 8.

$$\begin{cases} If & \kappa_2 \prec \kappa_3 \ then \\ & F = \left(\left[\eta^3_{(1,1:D)} \circ \left| \lambda^3_{(1,1:D)} \right| \right]' \times Q_{(1,1:N)} \right)' \\ else & \\ & F = \lambda^3_{(N,1)} \times Q_{(1,D)} \\ end & \end{cases} \tag{8}$$

Here, $\kappa_{2:3}, \eta$, and λ are random numbers that receive a new value in each call, where $\kappa_{2:3}$, $\eta \sim \mathbf{U}(0,1)$, $\lambda \sim \mathbf{N}(0,1)$, and $(\cdot,\cdot)$ sized *all-ones matrix* $Q_{(\cdot,\cdot)} = 1$.

BSD's trial *pattern vector* (*i.e.*, T_i) generation process is a *random crossover* process. In the BSD, the trial *pattern vectors* are generated by using the system equation defined in Eq 9.

$$T = P + F \circ M \circ \left((w^*)^3 \circ E + \left(1 - (w^*)^3\right) \circ bestP - P \right) \quad | \ w^*_{(1:N,1)} \sim \mathbf{U}(0,1) \tag{9}$$

where, $E = w \cdot P_{L_1} + (1-w) \cdot P_{L_2} \quad | \quad w_{(1:N,1:D)} \sim \mathbf{U}(0,1)$ and L_1 and L_2 are defined in Eq 10.

$$L_1 = permute(1:N), L_2 = permute(1:N) \ \mid \ L_1 \neq [1:N], \ L_1 \neq L_2 \tag{10}$$

If an individual of a trial *pattern vector* exceeds the search space, the individual is updated using the Eq 11.

$$If \ (T_{i,j} < low_j) \ or \ (T_{i,j} > up_j) \ then \ T_{i,j} = low_j + \delta \cdot (up_j - low_j) \tag{11}$$

Here, $\delta \sim \mathbf{U}(0,1)$.

The *objective function*, $\mathcal{F}(\cdot)$, values, *fitT*, of the trial *pattern vectors* are computed by using Eq 12;

$$fitT = \mathcal{F}(T) \tag{12}$$

Trial *pattern vector*, which provides a better objective function value than the corresponding *pattern vector*, is used to update the relevant *pattern vector*. It is also updated in the objective function value of the *pattern vector*. This process is achieved by using Eq 13.

$$If \ fitT_{(i^*)} < fitP_{(i^*)}, \ \ [P_{(i^*)}, \ fitP_{(i^*)}] = [T_{(i^*)}, fitT_{(i^*)}] \ | \ i^* \in [1:N] \tag{13}$$

In the present iteration step, the *pattern vector* which provides the best solution, *bestP*, and its objective function value, *solP*, are obtained by using Eq 14.

$$[solP, bestP] = [fitP_{(\gamma)}, P_{(\gamma)}] \ \mid \ fitP_{(\gamma)} = \min(fitP) \tag{14}$$

The pseudo-code of BSD is given in Fig. 1.

The similarities and differences of BSD and the tested methods are as follows:

- BSD's *random crossover* process differs from the corresponding *crossover* processes of the tested methods.
- The BSD's *crossover* process is a stochastic process based on the use of Bernstain polynomials and

Input: Objective Function: $\mathcal{F}$, Search-Space Limits: (low, up), Size of Pattern Matrix: N, Dimension of problem: D, Maximum Number of Iterations: *MaxCycle*
Output: $solP$: Global Minimum, $bestP$: Global Minimizer
`// Initialization`
$P_{i,j} \sim \mathbf{U}(low_j, up_j) \mid i = [1:N],\; j = [1:D]$, *where* $i, j \in \mathbb{Z}^+$
$fitP_i = \mathcal{F}(P_i)$
$[solP, bestP] = [fitP_{(\gamma)}, P_{(\gamma)}] \mid fitP_{(\gamma)} = \min(fitP) \mid \gamma \in [1:N]$
for *Iteration=1 to MaxCycle* **do**
  `// Generation of Mutation Control Matrix ; M`
  $M_{(i=1:N, j=1:D)} = 0$
  for *i=1 to N* **do**
    $u = permute(1:D)$
    Generate β, where $\beta \sim \mathbf{U}(0,1)$
    Generate κ_0, where $\kappa_0 = \left\lceil 3 \cdot {\kappa_1}^3 \right\rceil$, $\kappa_1 \sim U[0\ 1]$, $\kappa_0 \in \mathbf{U}\{1:3\}$
    switch κ_0 **do**
      case 1, $\rho = (1-\beta)^2$
      case 2, $\rho = 2 \cdot \beta \cdot (1-\beta)$
      case 3, $\rho = \beta^2$
    endsw
    $M_{(i, u(1:\lceil \rho \cdot D \rceil))} = 1$
  end
  `// Generation of Evolutionary Step Size; F`
  $\kappa_{2:3}, \eta$, and λ are random numbers, where $\kappa_{2:3} \sim \mathbf{U}(0,1)$, $\eta \sim \mathbf{U}(0,1)$, $\lambda \sim \mathbf{N}(0,1)$, and all-ones matrix $Q_{(\cdot,\cdot)} = 1$
  if $\kappa_2 < \kappa_3$ **then**
    $F = \left(\left[\eta^3_{(1,1:D)} \circ \left| \lambda^3_{(1,1:D)} \right| \right]' \times Q_{(1,1:N)} \right)'$
  else
    $F = \lambda^3_{(N,1)} \times Q_{(1,D)}$
  end
  `// Generation of Trial Pattern Vectors; T`
  $L_1 = permute(1:N)$, $L_2 = permute(1:N) \mid L_1 \neq [1:N]$, $L_1 \neq L_2$
  $E = w \cdot P_{L_1} + (1-w) \cdot P_{L_2} \mid w_{(1:N,1:D)} \sim \mathbf{U}(0,1)$
  $T = P + F \circ M \circ \left((w^*)^3 \circ E + \left(1 - (w^*)^3\right) \circ bestP - P \right) \mid w^*_{(1:N,1)} \sim \mathbf{U}(0,1)$
  `// Boundary Control Mechanism`
  if $(T_{i,j} < low_j)$ *or* $(T_{i,j} > up_j)$ **then** $T_{i,j} = low_j + \delta \cdot (up_j - low_j) \mid \delta \sim \mathbf{U}(0,1)$
  `// Update`
  $fitT = \mathcal{F}(T)$
  if $fitT_{(i^*)} < fitP_{(i^*)}$ **then** $[P_{(i^*)}, fitP_{(i^*)}] = [T_{(i^*)}, fitT_{(i^*)}] \mid i^* \in [1:N]$
  `// Get the solutions`
  $[solP, bestP] = [fitP_{(\gamma)}, P_{(\gamma)}] \mid fitP_{(\gamma)} = \min(fitP)$
end

Figure 1. Pseudo code of the Bernstein-Search Differential Evolution Algorithm (BSD). The unoptimized Matlab code of the BSD is publicly available at (Mathworks , 2019).

there is no parameter controlling this process.

- Since BSD uses the global minimizer *pattern vector* in its system equation (*i.e.*, Eq 9), it shows a partially elitist behavior whereas ABC and CUCKOO are elitist algorithms.
- The BSD is sensitive to the values of common control parameters (*i.e.*, N, D and number of iterations) such as tested methods (*i.e.*, ABC, JADE, CUCKOO and WDE).
- BSD can operate in parallel to calculate the objective function values, and BSD can perform *bounded / unbounded* search without any modification.
- BSD has a one-step search process, unlike ABC and CUCKOO.

2.1. Nomenclature

Symbol	Meaning / Definition
$\mathcal{F}$	Objective function.
low, up	Lower and upper limits of search-space.
N	Size of *pattern matrix*.
D	Dimension of problem.
$MaxCycle$	Maximum number of iterations.
$gmin$	Global minimum value.
$gbest$	The global minimizer *pattern vector*.
$\kappa_{(\cdot)} \sim U(0,1), \kappa_{(\cdot)} \neq 0$	κ is a uniform random number.
$\lambda_{(\cdot)} \sim N(0,1)$	λ is a normal random number.
$\eta \sim N(0,1)$	η is a normal random number.
$\beta \sim U(0,1)$	β is a uniform random number.
$U(\cdot)$	Continuous Uniform Distribution.
$U\{\cdot\}$	Discrete Uniform Distribution.
$P_{(i0,j0)} \mid P_{(i0,j0)} \sim \mathbf{U}(low_{(j0)}, up_{(j0)})$	Pattern vectors of pattern matrix.
$fitP_{(i0)}$	Fitness values of $P_{i0=1:N}$.
permute()	Permuting function.
$\circ$	Hadamart multiplication operator.

2.2. Bernstain Polynomials

The 2^{nd} degree Bernstain polynomials (Azhari & et al , 2018) are identified using Eq.s 15-16;

$$B_{s,n}(t) = \binom{n}{s} t^s (1-t)^{n-s} \tag{15}$$

Here $s = 0 : n$, $\binom{n}{s} = \frac{n!}{s!(n-s)!}$. Eq. 16 generates $(n+1)$ sized n^{th} degree Bernstain polynomials. For $s < 0$ and $s > n$, $B_{s,n} = 0$.

$$\begin{aligned} B_{0,2}(t) &= (1-t)^2 \\ B_{1,2}(t) &= 2t(1-t) \\ B_{2,2}(t) &= t^2 \end{aligned} \tag{16}$$

Fig. 2 illustrates 2^{nd} degree Bernstain polynomials for $0 \leq t \leq 1$;

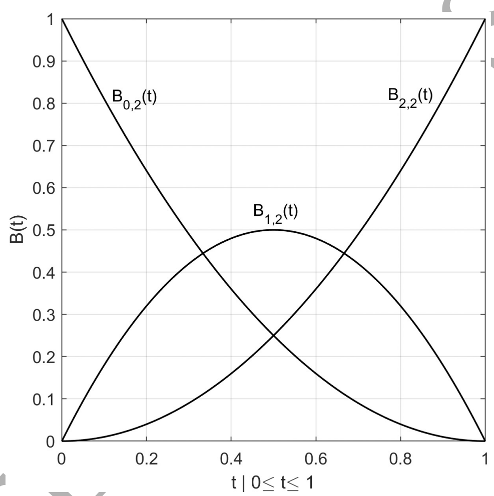


Figure 2. 2^{nd} degree Bernstain polynomials.

2.3. Benchmark Problems

Numerical optimization problem solving capability of BSD that is introduced in this paper is examined by using 30 benchmark problems of CEC'2014 (Liang, Qu, & Suganthan , 2013), F1-F30, and 60 classic benchmark problems (Matlab , 2019a,b), F31-F90. Structures of F1-F30 are more complex than those of the classic benchmark problems and their solutions ares relatively more difficult. Dimensions of F1-F30 are selected as 10. Detailed mathematical definitions belonging to F1-F30 can be found in (Liang, Qu, & Suganthan , 2013). Some properties of the classic benchmark problems are given in Table 1.

In this paper, in order to examine the success of BSD in the solution of *real-world* engineering problems, an *image vectorization problem* (Bergen & Ross , 2012) for 12 images (*i.e.*, Test Img.s 1-12) and one *TIN refinement problem* (Chernikov, & Chrisochoides , 2012; Civicioglu, & Alci , 2004) for Mount Erciyes, in Kayseri City in Turkey, were used;

2.4. Statistical Analysis

In this paper, a two-tailed Wilcoxon Signed Rank Test (Derrac, Garca, & Molina , 2011; Civicioglu , 2013a, 2012, 2013b) was used for the statistical comparison of the results obtained from the experiments as in (Civicioglu, Besdok, & et al , 2018). In statistical comparisons, the level of significance is set to 0.05 (Derrac, Garca, & Molina , 2011).

3. Experiments

The related benchmark problems were solved using 50 different initial *pattern matrix*. Each algorithm used the same initial *pattern matrix* at each experiment. In experiments performed, the numerical resolution level is 10^{-16}. Dimension of *pattern matrix* was set to 30 in the experiments. Experimental test results were recorded at the end of the 200,000th iterations. In this paper, Mersenne Twister was used as the random number generator (Matsumoto, & Nishimura , 1998). Initial values of control parameters of the tested methods used in this paper are given in Table 2.

3.1. Numerical function optimization

In this section, success of BSD in numerical function optimization problems was examined with detailed applications.

The mean value, *Mu*, and standard deviation value, *Std*, of the solutions obtained by BSD and test methods were calculated, in order to conventionally analyze the ability of the related methods to reach the minimum of the related problem. Conventional statistical results (*i.e.*, *Mu* and *Std*) are given in Tables 3-5.

The Wilcoxon Signed rank test ($p = 0.05$) based statistical comparison results of the numerical problem solving success of BSD and tested methods for F1-F90 are given in Tables 6-7.

On the last row of Table 6, results obtained from BSD and tested methods were compared in (+,=,-). (+) is the benchmark function number that BSD obtains a statistically better result than the related tested method. (=) is the benchmark function number that the performances of BSD and the related tested method are statistically equal. (-) is the benchmark function number that the related tested method obtained a statistically better result than BSD.

Table 1. The classic benchmark problems (Low,Up; Lower and upper limits of search-space, Dim: Dimension of problem).

#	Function	Low	Up	Dim	#	Function	Low	Up	Dim	#	Function	Low	Up	Dim
F31	Absolute	-100	100	30	F51	Hartman3	0	1	3	F71	Rosenbrock	-30	30	30
F32	Ackley	-32	32	30	F52	Hartman6	0	1	6	F72	Schaffer	-100	100	2
F33	Beale	-4.5	4.5	5	F53	Himmelblau	-5	5	30	F73	Schwefel	-500	500	30
F34	Bohachecsky1	-100	100	2	F54	Hump	-5	5	2	F74	Schwefel_1_2	-100	100	30
F35	Bohachecsky2	-100	100	2	F55	Kowalik	-5	5	4	F75	Schwefel_2_22	-10	10	30
F36	Bohachecsky3	-100	100	2	F56	Langermann	0	10	2	F76	Shekel10	0	10	4
F37	Booth	-10	10	2	F57	Langermann	0	10	5	F77	Shekel5	0	10	4
F38	Branin	-5	10	2	F58	Langermann	0	10	10	F78	Shekel7	0	10	4
F39	Colville	-10	10	4	F59	Levy	-10	10	30	F79	Shubert	-10	10	2
F40	Dixonprice	-10	10	30	F60	Matyas	-10	10	2	F80	Sixhumpcamelback	-5	5	2
F41	Dropwave	-2	2	2	F61	Michalewics10	0	pi	10	F81	Solomon	-100	100	30
F42	Easom	-100	100	2	F62	Michalewics2	0	pi	2	F82	Sphere2	-100	100	30
F43	Eggholder	-512	512	2	F63	Michalewics5	0	pi	5	F83	Step2	-100	100	30
F44	Fletcher	-pi	pi	2	F64	Penalized	-50	50	30	F84	Stepint	-5.12	5.12	5
F45	Fletcher	-pi	pi	5	F65	Penalized2	-50	50	30	F85	Sumsquares	-10	10	30
F46	Fletcher	-pi	pi	10	F66	Perm	-4	4	4	F86	Trid	-36	36	6
F47	Foxholes	-65.536	65.536	2	F67	Powell	-4	5	24	F87	Trid	-100	100	10
F48	Giunta	-1	1	2	F68	Powersum	0	4	4	F88	Weierstrass	-0.5	0.5	30
F49	Goldsteinprice	-2	2	2	F69	Quartic	-1.28	1.28	30	F89	Whitley	-10	10	30
F50	Griewank	-600	600	30	F70	Rastrigin	-5.12	5.12	30	F90	Zakharov	-5	10	10

Table 2. Initial values of control parameters of the tested methods.

#	Algorithm	Initial Values of Control Parameters
1	ABC	$limit = N \cdot D$ $Size\,of\,employed\,bee = (size\,of\,colony)/2$
2	JADE	$F \sim N(\mu_c, 0.10)$ $CR \sim Cauchy(\mu_F, 0.10)$, $c = 0.10$, $p = 0.05$
3	CUCKOO	$\beta = 1.50$, $p_0 = 0.25$
4	WDE	There are no parameters other than the common parameters (*i.e.*, the number of iterations, N, and D).
5	BSD	There are no parameters other than the common parameters (*i.e.*, the number of iterations, N, and D).

Table 3. Results belonging to tests carried out by using CEC'2014 benchmark problems (*i.e.*, F1-F30).

Fnc	ABC		JADE		CUCKOO		WDE		BSD	
	Mu	Std	Mu	Std	Mu	Std	Mu	Std	Mu	Std
F1	1.144E+04	1.008E+03	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	1.536E-03	1.354E-04
F2	7.982E-02	7.371E-03	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F3	9.535E-01	3.159E-03	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F4	2.363E-03	6.489E-05	3.478E+01	9.552E-01	0.000E+00	0.000E+00	0.000E+00	0.000E+00	3.638E-03	9.991E-05
F5	4.228E+00	1.831E-01	1.335E-01	5.780E-03	2.000E+01	8.662E-01	1.999E+01	8.658E-01	1.901E+01	8.232E-01
F6	9.143E-01	5.473E-02	0.000E+00	0.000E+00	0.000E+00	0.000E+00	1.274E+00	7.625E-02	0.000E+00	0.000E+00
F7	1.137E-12	2.996E-15	0.000E+00	0.000E+00	7.396E-03	1.949E-05	5.161E-11	1.360E-13	6.049E-03	1.594E-05
F8	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F9	3.997E+00	3.206E-01	0.000E+00	0.000E+00	1.990E+00	1.596E-01	4.975E+00	3.990E-01	1.772E+00	1.421E-01
F10	8.185E-12	4.362E-13	2.498E-01	1.331E-02	6.245E-02	3.328E-03	1.644E-07	8.759E-09	0.000E+00	0.000E+00
F11	4.329E+01	1.595E+00	6.245E-02	2.302E-03	2.689E+01	9.909E-01	3.524E+01	1.299E+00	3.123E-01	1.151E-02
F12	1.339E-01	7.104E-03	4.323E-02	2.294E-03	2.108E-03	1.118E-04	1.931E-02	1.025E-03	1.108E-04	5.878E-06
F13	7.443E-02	1.377E-03	6.994E-02	1.294E-03	1.095E-01	2.026E-03	7.638E-02	1.413E-03	7.367E-02	1.363E-03
F14	1.063E-01	9.430E-03	1.028E-01	9.126E-03	4.467E-02	3.965E-03	5.834E-02	5.178E-03	8.822E-02	7.829E-03
F15	4.679E-01	1.688E-02	3.847E-01	1.388E-02	5.175E-01	1.867E-02	4.250E-01	1.533E-02	4.612E-01	1.664E-02
F16	1.510E+00	2.417E-02	4.596E-01	7.352E-03	1.178E+00	1.884E-02	1.554E+00	2.486E-02	1.046E+00	1.673E-02
F17	3.431E+04	9.474E+02	1.351E+02	3.729E+00	0.000E+00	0.000E+00	4.816E+00	1.330E-01	4.171E+01	1.152E+00
F18	6.938E+00	3.552E-02	1.650E-01	8.444E-04	7.014E-05	3.591E-07	2.769E-02	1.417E-04	4.275E-04	2.189E-06
F19	1.188E-01	1.090E-02	0.000E+00	0.000E+00	0.000E+00	0.000E+00	2.188E-01	2.007E-02	9.937E-03	9.115E-04
F20	4.789E+00	1.589E-01	3.162E-02	1.049E-03	3.894E-06	1.292E-07	4.943E-02	1.640E-03	2.608E-02	8.653E-04
F21	9.519E+02	2.905E+01	2.419E-01	7.380E-03	1.189E-05	3.629E-07	1.111E-02	3.390E-04	1.677E+01	5.117E-01
F22	1.436E-01	1.338E-02	3.123E-01	2.909E-02	9.608E-03	8.950E-04	7.901E-03	7.360E-04	3.764E-03	3.507E-04
F23	9.434E+00	6.494E-03	3.295E+02	2.268E-01	3.295E+02	2.268E-01	3.295E+02	2.268E-01	3.259E+02	2.244E-01
F24	1.145E+02	2.428E+00	1.087E+02	2.307E+00	1.096E+02	2.324E+00	1.091E+02	2.314E+00	1.073E+02	2.277E+00
F25	1.160E+02	9.900E+00	1.103E+02	9.416E+00	1.000E+02	8.537E+00	1.193E+02	1.018E+01	1.084E+02	9.257E+00
F26	8.838E+01	4.347E+00	1.000E+02	4.920E+00	1.001E+02	4.922E+00	1.000E+02	4.919E+00	9.707E+01	4.774E+00
F27	6.608E+00	1.473E-01	4.001E+02	8.921E+00	6.957E-01	1.551E-02	2.372E+00	5.287E-02	1.796E+00	4.004E-02
F28	1.673E+02	1.638E+01	4.780E+02	4.680E+01	3.568E+02	3.494E+01	3.570E+02	3.496E+01	3.542E+02	3.468E+01
F29	2.381E+02	2.310E+01	2.242E+02	2.175E+01	1.000E+02	9.702E+00	1.020E+02	9.895E+00	2.260E+02	2.192E+01
F30	4.871E+02	1.468E+01	4.657E+02	1.403E+01	4.623E+02	1.393E+01	2.345E+02	7.066E+00	3.571E+02	1.076E+01

Table 4. Results belonging to tests carried out by using classic benchmark problems (*i.e.*, F31-F60).

Fnc	ABC		JADE		CUCKOO		WDE		BSD	
	Mu	Std	Mu	Std	Mu	Std	Mu	Std	Mu	Std
F31	3.851E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F32	2.931E-14	2.247E-15	6.573E-15	1.740E-15	7.283E-15	1.421E-15	7.283E-15	1.421E-15	7.994E-15	0.000E+00
F33	1.511E-11	1.631E-11	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F34	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F35	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F36	1.366E-15	1.064E-15	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F37	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F38	3.979E-01	0.000E+00	3.979E-01	0.000E+00	3.979E-01	0.000E+00	3.979E-01	0.000E+00	3.979E-01	0.000E+00
F39	3.268E-02	1.655E-02	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F40	4.654E-15	1.914E-15	6.667E-01	0.000E+00	6.667E-01	0.000E+00	6.667E-01	0.000E+00	6.667E-01	0.000E+00
F41	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F42	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F43	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F44	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F45	1.513E-02	2.760E-02	3.719E+01	7.438E+01	0.000E+00	0.000E+00	1.702E-15	8.484E-16	0.000E+00	0.000E+00
F46	5.104E+00	1.827E+00	3.130E+02	6.073E+02	0.000E+00	0.000E+00	3.196E-06	3.459E-06	0.000E+00	0.000E+00
F47	9.980E-01	0.000E+00	9.980E-01	0.000E+00	9.980E-01	0.000E+00	9.980E-01	0.000E+00	9.980E-01	0.000E+00
F48	6.447E-02	0.000E+00	6.447E-02	0.000E+00	6.447E-02	0.000E+00	6.447E-02	0.000E+00	6.447E-02	0.000E+00
F49	3.000E+00	1.376E-15	3.000E+00	0.000E+00	3.000E+00	1.986E-16	3.000E+00	0.000E+00	3.000E+00	1.986E-16
F50	0.000E+00	0.000E+00	3.451E-03	4.297E-03	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F51	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F52	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F53	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F54	4.651E-08	0.000E+00	4.651E-08	0.000E+00	4.651E-08	0.000E+00	4.651E-08	0.000E+00	4.651E-08	0.000E+00
F55	3.620E-04	4.423E-05	3.075E-04	0.000E+00	3.075E-04	0.000E+00	3.075E-04	0.000E+00	3.075E-04	0.000E+00
F56	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F57	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F58	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F59	3.696E-16	0.000E+00	3.581E-02	4.386E-02	1.088E-01	2.175E-01	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F60	5.287E-16	5.491E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00

Table 5. Results belonging to tests carried out by using classic benchmark problems (*i.e.*, F61-F90).

Fnc	ABC		JADE		CUCKOO		WDE		BSD	
	Mu	Std	Mu	Std	Mu	Std	Mu	Std	Mu	Std
F61	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F62	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F63	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F64	3.128E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F65	3.441E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F66	1.335E-02	8.154E-03	1.813E-03	2.195E-03	2.045E-07	4.091E-07	7.760E-06	7.352E-06	0.000E+00	0.000E+00
F67	1.823E-04	2.321E-05	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	5.644E-11	4.968E-11
F68	1.048E-03	3.846E-04	1.727E-04	2.358E-04	0.000E+00	0.000E+00	9.151E-09	7.745E-09	2.044E-07	3.331E-07
F69	3.615E-02	8.575E-03	1.799E-03	8.219E-04	2.887E-04	1.025E-04	2.862E-04	8.875E-05	1.920E-04	5.237E-05
F70	0.000E+00	0.000E+00	0.000E+00	0.000E+00	7.960E-01	1.160E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F71	2.762E-02	2.134E-02	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F72	9.406E-09	1.480E-08	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F73	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F74	1.168E-01	2.078E-01	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F75	3.615E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F76	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F77	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F78	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F79	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F80	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F81	6.999E-01	8.944E-02	3.799E-01	1.327E-01	2.399E-01	4.899E-02	1.999E-01	6.325E-02	1.761E-01	5.573E-02
F82	4.222E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F83	0.000E+00	0.000E+00	2.000E-01	4.000E-01	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F84	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F85	3.213E-16	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F86	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F87	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F88	0.000E+00	0.000E+00	2.026E-01	4.052E-01	2.331E-02	4.617E-02	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F89	7.026E+01	4.995E+01	1.298E+02	9.694E+01	2.335E+01	1.757E+01	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F90	1.136E-04	1.376E-04	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00

Table 6. Comparison of CEC'2014 benchmark problems (i.e., F1-F30) solving successes of BSD and tested methods by using Wilcoxon Signed rank test (p=0.05).

Fnc	ABC				JADE				CUCKOO				WDE			
	p	R+	R-	stat.	p	R+	R-	stat.	p	R+	R-	stat.	p	R+	R-	stat.
F1	9.13E-07	0	465	+	0.999999	465	0	-	0.999999	465	0	-	0.999999	465	0	-
F2	9.13E-07	0	465	+	1	0	0	=	1	0	0	=	1	0	0	=
F3	9.13E-07	0	465	+	1	0	0	=	1	0	0	=	1	0	0	=
F4	0.999999	465	0	-	9.13E-07	0	465	+	0.999999	465	0	-	0.999999	465	0	-
F5	0.999999	465	0	-	0.999999	465	0	-	0.000345	67	398	+	0.000371	68	397	+
F6	9.13E-07	0	465	+	1	0	0	=	1	0	0	=	9.13E-07	0	465	+
F7	0.999999	465	0	-	0.999999	465	0	-	9.13E-07	0	465	+	0.999999	465	0	-
F8	1	0	0	=	1	0	0	=	1	0	0	=	1	0	0	=
F9	9.13E-07	0	465	+	0.999999	465	0	-	1.03E-05	25	440	+	9.13E-07	0	465	+
F10	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+
F11	9.13E-07	0	465	+	0.999999	465	0	-	9.13E-07	0	465	+	9.13E-07	0	465	+
F12	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+
F13	0.013875	125	340	+	0.999999	465	0	-	9.13E-07	0	465	+	2.48E-06	10	455	+
F14	2.04E-06	8	457	+	7.82E-06	22	443	+	0.999999	465	0	-	0.999999	465	0	-
F15	0.072097	161	304	=	0.999999	465	0	-	9.13E-07	0	465	+	0.999998	455	10	-
F16	9.13E-07	0	465	+	0.999999	465	0	-	9.13E-07	0	465	+	9.13E-07	0	465	+
F17	9.13E-07	0	465	+	9.13E-07	0	465	+	0.999999	465	0	-	0.999999	465	0	-
F18	9.13E-07	0	465	+	9.13E-07	0	465	+	0.999999	465	0	-	9.13E-07	0	465	+
F19	9.13E-07	0	465	+	0.999999	465	0	-	0.999999	465	0	-	9.13E-07	0	465	+
F20	9.13E-07	0	465	+	9.13E-07	0	465	+	0.999999	465	0	-	9.13E-07	0	465	+
F21	9.13E-07	0	465	+	0.999999	465	0	-	0.999999	465	0	-	0.999999	465	0	-
F22	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+
F23	0.999999	465	0	-	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+
F24	9.13E-07	0	465	+	0.025351	137	328	+	0.000499	72	393	+	0.00679	112	353	+
F25	0.005705	109	356	+	0.261861	201	264	=	0.999379	389	76	-	0.000218	61	404	+
F26	0.999992	442	23	-	0.020862	133	332	+	0.069314	160	305	=	0.010621	120	345	+
F27	9.13E-07	0	465	+	9.13E-07	0	465	+	0.999999	465	0	-	9.13E-07	0	465	+
F28	0.999999	465	0	-	1.01E-06	1	464	+	0.255209	200	265	=	0.325453	210	255	=
F29	0.056624	155	310	=	0.5	232	233	=	0.999999	465	0	-	0.999999	465	0	-
F30	9.13E-07	0	465	+	9.13E-07	0	465	+	9.13E-07	0	465	+	0.999999	465	0	-
+				21				14				13				17
-				6				10				11				9
=				3				6				6				4

+: BSD is winner, −: tested method is winner, =: Similar performance.

Table 7. Comparison of classic benchmark problems (i.e., F31-F90) solving successes of BSD and tested methods by using Wilcoxon Signed rank test (p=0.05).

Fnc	ABC				JADE				CUCKOO				WDE			
	p	R+	R-	stat.	p	R+	R-	stat.	p	R+	R-	stat.	p	R+	R-	stat.
F31	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F32	3.94E-02	15	0	+	1.57E-01	0	3	-	3.17E-01	0	1	-	3.17E-01	0	1	-
F33	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F34	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F35	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F36	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F37	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F38	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F39	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F40	4.22E-02	0	15	-	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F41	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F42	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F43	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F44	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F45	4.31E-02	15	0	+	3.17E-01	1	0	+	1.00E+00	0	0	=	4.31E-02	15	0	+
F46	4.31E-02	15	0	+	1.09E-01	6	0	+	1.00E+00	0	0	=	4.31E-02	15	0	+
F47	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F48	3.17E-01	0	1	-	5.64E-01	4	2	-	8.33E-02	0	6	-	8.33E-02	0	6	-
F49	3.84E-02	15	0	+	2.53E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=
F50	3.17E-01	1	0	+	1.80E-01	3	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=
F51	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F52	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F53	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F54	1.00E+00	2	2	-	3.17E-01	0	1	-	3.17E-01	0	1	-	4.55E-02	10	0	+
F55	4.31E-02	15	0	+	4.55E-02	10	0	+	1.57E-01	3	0	+	8.33E-02	6	0	+
F56	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F57	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F58	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F59	4.31E-02	15	0	+	1.57E-01	3	0	+	3.17E-01	1	0	+	1.00E+00	0	0	=
F60	1.09E-01	6	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F61	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F62	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F63	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F64	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F65	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F66	4.31E-02	15	0	+	6.79E-02	10	0	+	3.17E-01	1	0	+	4.31E-02	15	0	+
F67	4.31E-02	15	0	+	4.31E-02	0	15	-	4.31E-02	0	15	-	4.31E-02	0	15	-
F68	4.31E-02	15	0	+	2.25E-01	12	3	-	4.31E-02	0	15	-	4.31E-02	0	15	-
F69	4.31E-02	15	0	+	4.31E-02	15	0	+	4.31E-02	15	0	+	4.31E-02	15	0	+
F70	1.00E+00	0	0	=	1.00E+00	0	0	=	1.80E-01	3	0	+	1.00E+00	0	0	=
F71	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F72	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F73	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F74	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F75	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F76	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F77	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F78	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F79	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F80	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F81	4.31E-02	15	0	+	4.31E-02	15	0	+	4.31E-02	15	0	+	3.94E-02	15	0	+
F82	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F83	1.00E+00	0	0	=	3.17E-01	1	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=
F84	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F85	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F86	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F87	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
F88	1.00E+00	0	0	=	3.17E-01	1	0	+	1.80E-01	3	0	+	1.00E+00	0	0	=
F89	4.31E-02	15	0	+	4.31E-02	15	0	+	4.22E-02	15	0	+	1.00E+00	0	0	=
F90	4.31E-02	15	0	+	1.00E+00	0	0	=	1.00E+00	0	0	=	1.00E+00	0	0	=
+				27				12				8				7
=				30				43				47				49
-				3				5				5				4

+: BSD is winner, −: tested method is winner, =: Similar performance.

In the solution of CEC'2014 benchmark problems, when results of BSD and tested methods were examined in (+,=,-) format, the following results are obtained; ABC (21,6,3), JADE (14,10,6), CUCKOO (13,11,6), WDE (17,9,4).

Accordingly, BSD has statistically better results (54.17%) than those of the tested methods in 65 out of a total of 120 piecewise comparisons. Successes of BSD and tested methods are statistically similar in 36 (30%) comparisons. Tested methods achieve statistically better results than BSD only in 19 comparisons (15.83%).

Results belonging to the comparison of classic benchmark problems (*i.e.*, F31-F90) solving successes of BSD and tested methods by using Wilcoxon Signed rank test ($p = 0.05$) are given in Table 7.

In the solution of classic benchmark problems, when results of BSD and tested methods were examined in (+,=,-) format, the following results are obtained; ABC (27,30,3), JADE (12,43,5), CUCKOO (8,47,5), WDE(7,49,4).

Accordingly, BSD has statistically better results (22.50%) than those of the tested methods in 54 out of a total of 240 piecewise comparisons. Successes of BSD and tested methods are statistically similar in 169 (70.42%) comparisons. Tested methods achieve statistically better results than BSD only in 17 comparisons (7.08%) for the classic benchmark problems.

In order to analyze the time-complexity of BSD and the tested methods, the mean runtime values that spent to converge to the results of the relevant benchmark problems were used. The mean runtime values of the BSD and the related tested methods to the solutions of the ralated benchmark problems are illustrated in Fig.s 3-4 as seconds. When examining Fig.s 3-4, it can be said that the fastest algorithms are JADE, CUCKOO, BSD, WDE and ABC to solve F1-F90. In general, the time complexity values of BSD and CUCKOO are similar and they are better than those of the ABC and WDE for the F1-F90.

Conventional benchmark problems are relatively easy to solve when compared to CEC2014 benchmark problems. Therefore, the success of algorithms used in solving classic benchmark problems is relatively similar.

3.2. Image Evolution Problem

Vector-geometry images (Bergen & Ross , 2012; Civicioglu, & Alci , 2004; Besdok, Civicioglu, & Alci , 2004) consist of discrete geometric shapes such as circles, lines, and ellipses. Vector-geometry images are used in graphical design, cartographic scientific visualization in Geomatics, and computer art applications. If the number of geometric shapes that make up the vector-geometry image can be limited, the stylized image of the target image can be generated. In the experiments performed in this section, vector-graphics images of test pictures (*i.e.*, related stylized images) are produced using circles and ellipses. The ellipses are generated by rotating a circle on the image canvas in 3D space. In each image evolution iteration step, only one circle is placed on the image canvas. The center;(x_0,y_0), radius;(*radii*), 3D turning angles; (Euler angles:$\omega_{euler}, \varphi_{euler}, \kappa_{euler}$), and alpha value, *alpha*, of each circle are optimized by using ABC, JADE, CUCKOO, WDE and BSD. Therefore, in experiments performed for *alpha* <1, each *pattern vector* has 7 individuals; x_0, y_0, *radii*, $\omega_{euler}, \varphi_{euler}, \kappa_{euler}$, and *alpha*. In the experiments performed for $alpha = 1$, each *pattern vector* has only 6 individuals; x_0, y_0, *radii*, $\omega_{euler}, \varphi_{euler}$, and κ_{euler}. In solving this problem, the size of *pattern matrix* and the maximum number of iterations were determined as 5 and 1000 for *alpha*<1, respectively. Similarly, the size of *pattern matrix* and the maximum number of iterations were determined as 5 and 100 for *alpha*=1, respectively. Fig. 5 shows a few steps of the transparent image evolution process, *i.e.*, *alpha*<1, of Test Img-5 by using BSD.

Each stylized image is obtained at the end of an image evolution process. The initial form of stylized images is a black background where the value of each pixel is zero. Then in each iteration, a new circle with optimized geometric parameters is placed on the image's canvas. Objective Function of image evolution problem has been given in Eq. 17 ;

$$\underset{I_{Evolved}}{argmin} \ \frac{1}{256^2}\sum(I_{Reference}-I_{Evolved}) \tag{17}$$

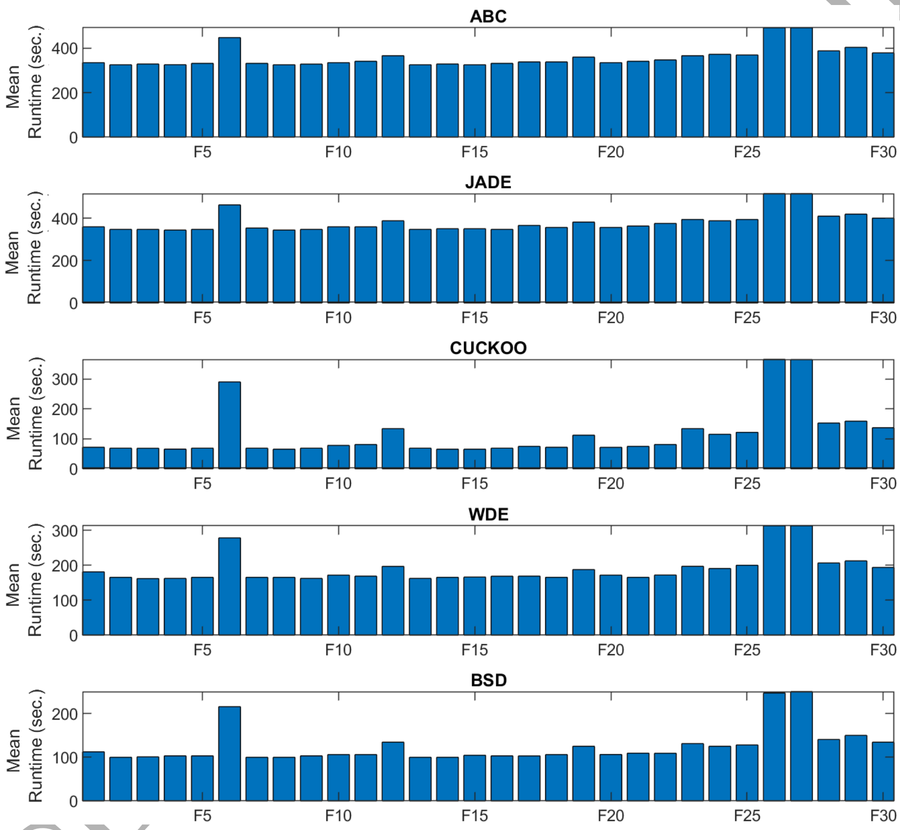


Figure 3. Analysis of time-complexities of related methods for F1-F30 as mean runtime values.

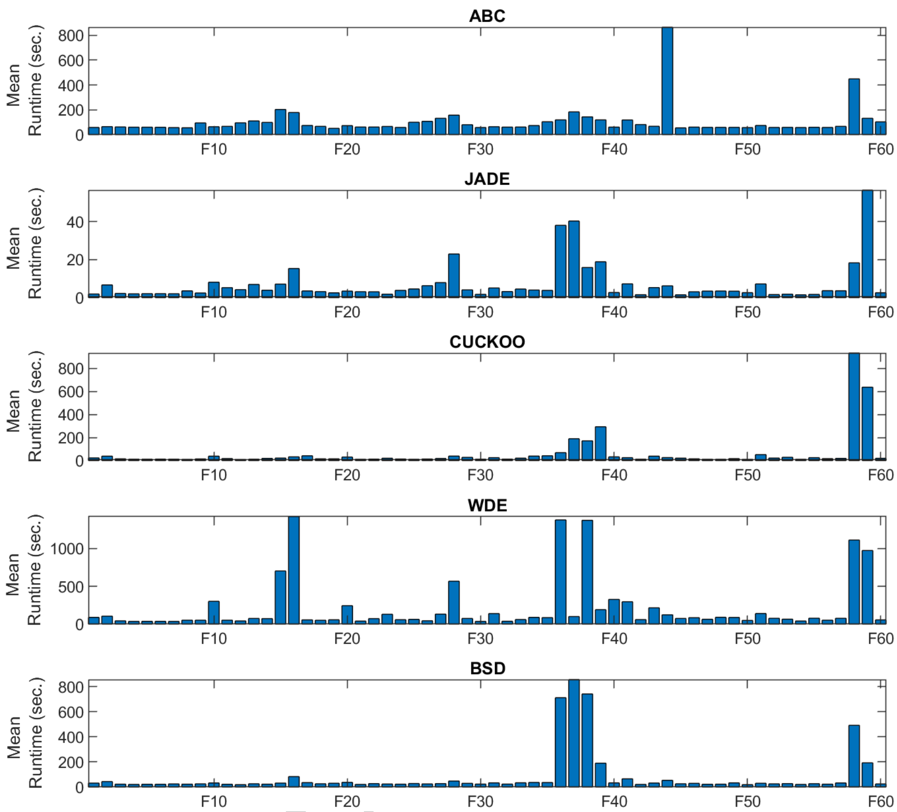


Figure 4. Analysis of time-complexities of related methods for F31-F90 as mean runtime values.

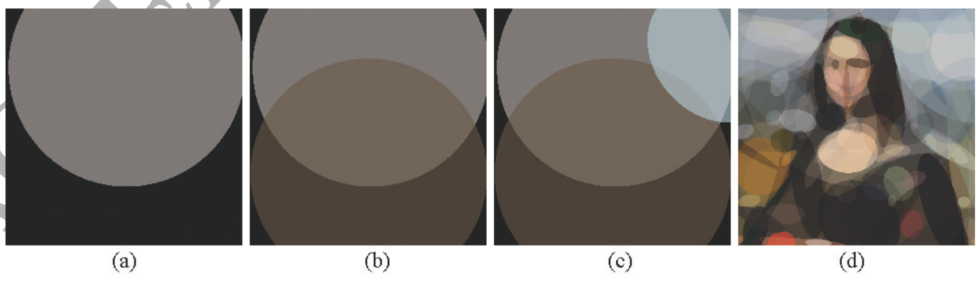


Figure 5. Process during image evolution for Test Img-5: (a) 1^{st} step, (b) 2^{nd} step, (c) 3^{rd} step, (d) 100^{th} step.

Here $I_{Reference}$ and $I_{Evolved}$ are 8-bit [256 256] pixels sized RGB images. $I_{Reference}$ and $I_{Evolved}$ denote original image and synthesized image by image evolution, respectively.

In this section, 12 test pictures (*i.e.*, Test Img.s 1-12) were used in the experiments performed. Relevant tests were performed using the *alpha* value and without the *alpha* value. *alpha* value controls the transparency of the geometric shapes used in the evolved image.

Fig.s. 6-7 visualize the solutions obtained for $alpha<1$. The synthetic images produced at this stage consist of 100 optimized circles.

Fig.s. 8-9 illustrate the results obtained for $alpha=1$ (*i.e.*, geometric shapes are opaque). The synthetic images produced at this stage consist of 1000 optimized circles. Objective function values which were obtained in the tests performed for $alpha<1$ and $alpha=1$ are given in Table 8 and Table 9, respectively. When Tables 8-9 are examined, it can be said that BSD generally achieves better objective function values than those of the tested methods. When Fig.s. 6-9 are examined, BSD achieves qualitatively good results, compared to comparison EAs. The results of the JADE are generally very close to the results of BSD. But JADE's computational load is too much higher than that of BSD. JADE also has a much more complex structure than that of BSD.

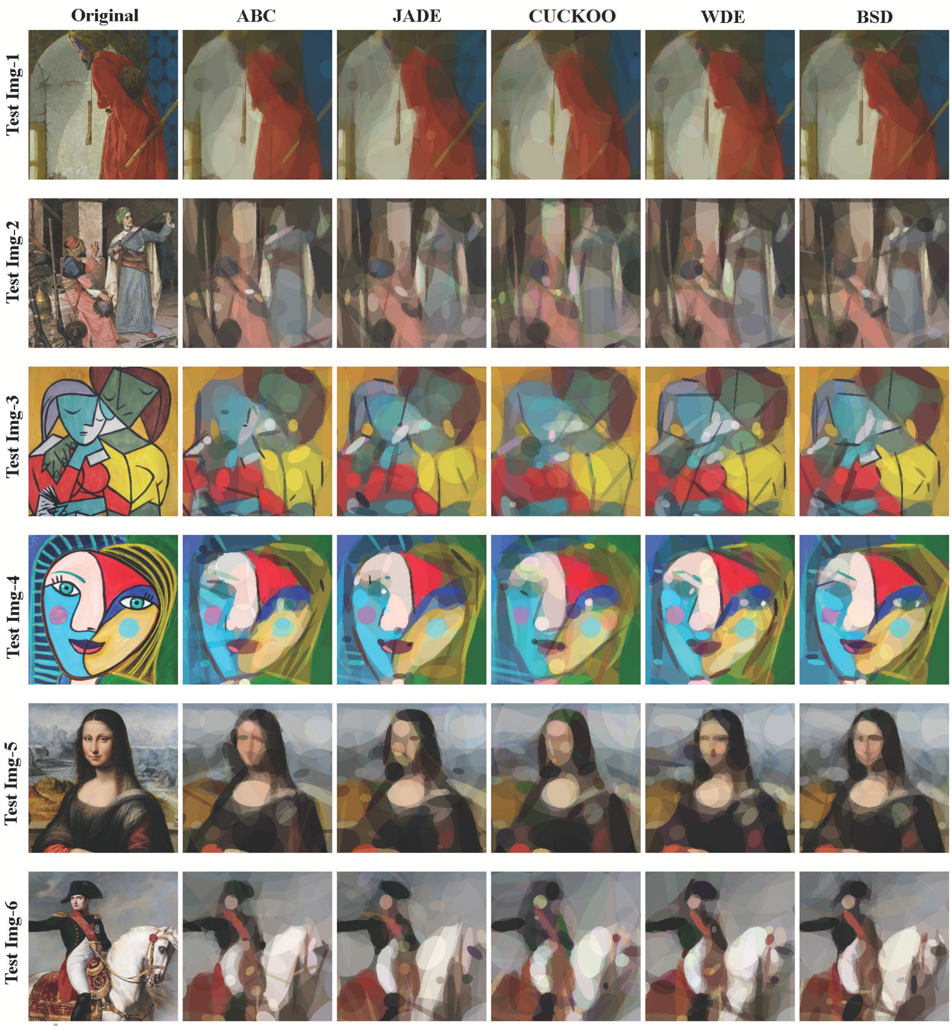


Figure 6. Visual results of solutions of image evolution problem for Test Img.s 1-6.

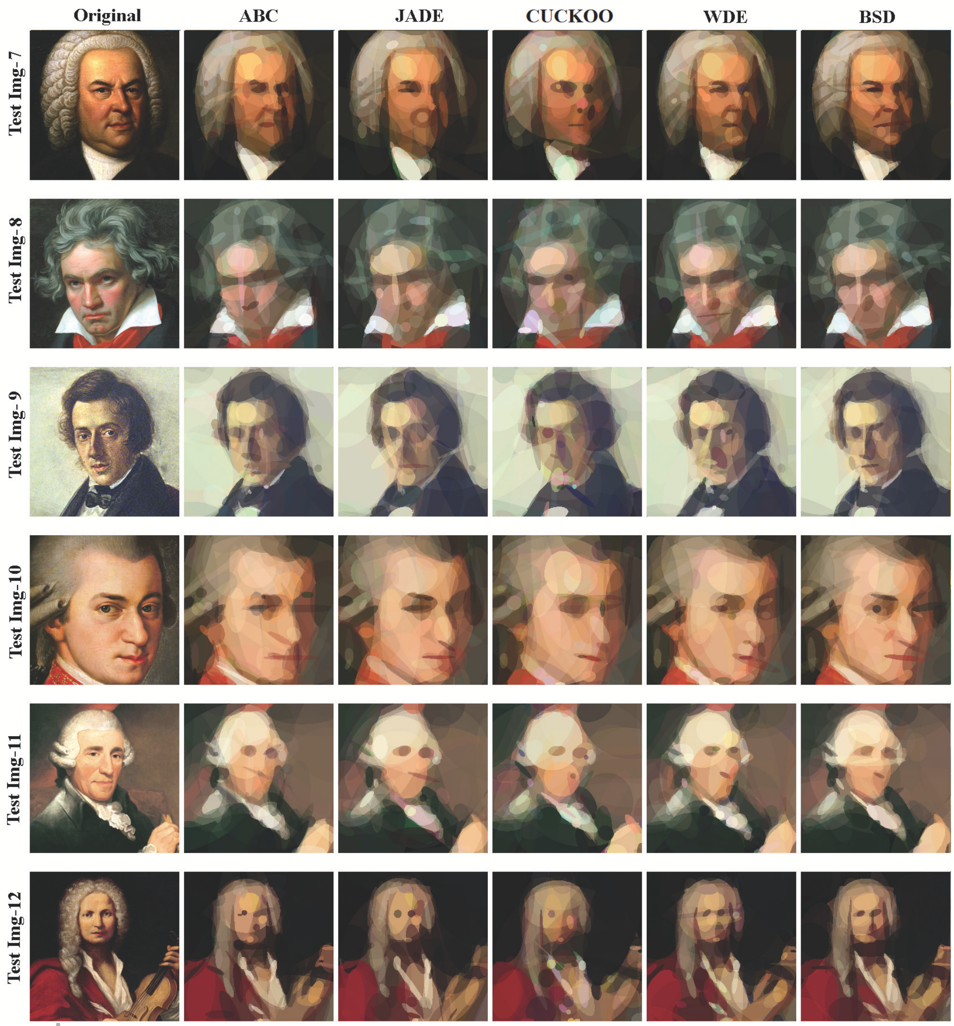


Figure 7. Visual results of solutions of image evolution problem for Test Img.s 7-12.

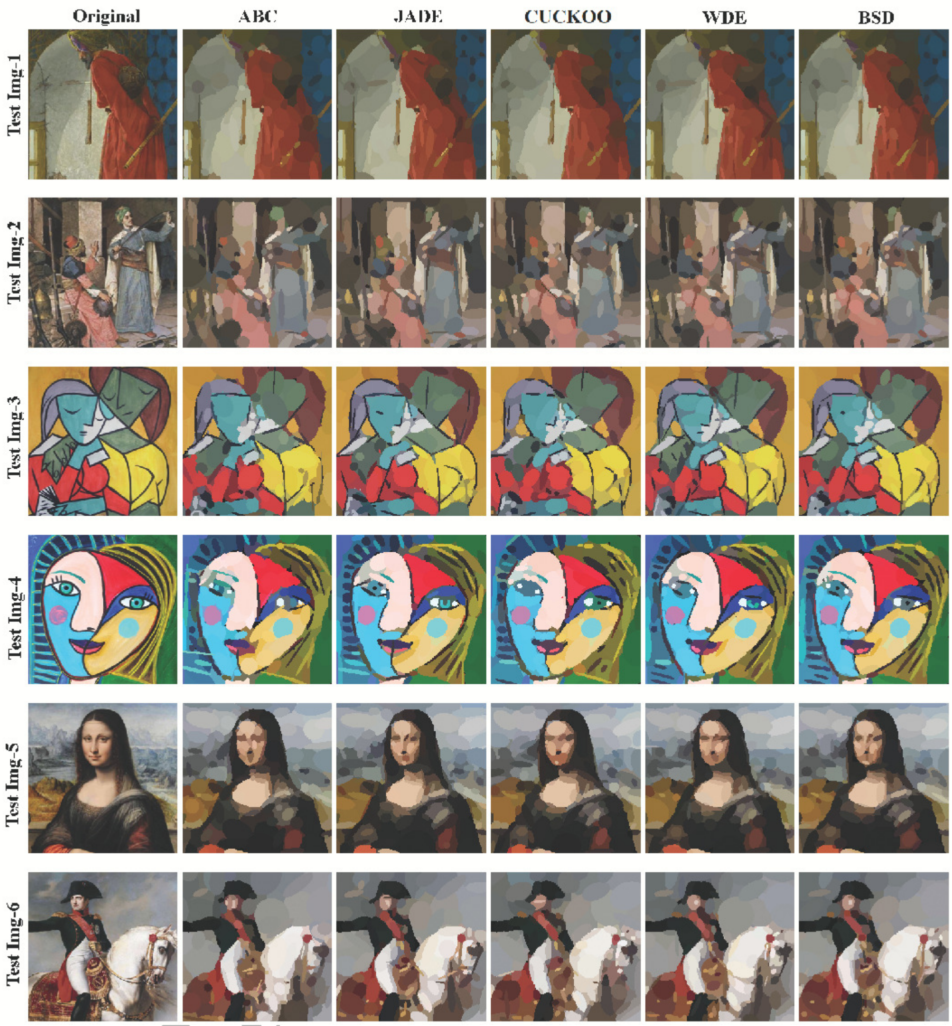


Figure 8. Visualization of the best image vectorization solutions obtained by the related EAs for the Test Img.s 1-6.

Figure 9. Visualization of the best image vectorization solutions obtained by related EAs for the Test Img.s 7-12.

Table 8. The objective function values obtained by solving the image vectorization problem when *alpha*<1.

Test Image	Algorithms				
	ABC	JADE	CUCKOO	WDE	BSD
Test Img-1	177.971	197.039	216.653	190.828	165.168
Test Img-2	551.371	553.867	633.186	537.566	520.017
Test Img-3	1066.319	1061.776	1248.424	997.936	912.446
Test Img-4	1645.673	1556.862	1865.778	1548.464	1434.537
Test Img-5	352.475	361.429	410.545	345.855	304.481
Test Img-6	643.089	639.979	735.660	595.416	571.776
Test Img-7	293.019	337.384	370.642	293.412	266.948
Test Img-8	352.995	384.919	432.819	322.157	297.737
Test Img-9	384.912	409.755	457.299	378.884	341.725
Test Img-10	262.337	263.699	315.934	248.989	223.538
Test Img-11	289.787	306.485	348.791	261.726	236.767
Test Img-12	317.463	332.101	402.183	304.511	264.368

Table 9. The objective function values obtained by solving the image vectorization problem when *alpha*=1.

Test Image	Algorithms				
	ABC	JADE	CUCKOO	WDE	BSD
Test Img-1	153.92	126.21	147.20	130.45	130.03
Test Img-2	458.26	400.87	435.71	413.71	398.24
Test Img-3	782.89	651.77	764.88	641.91	650.17
Test Img-4	1245.51	955.97	1142.16	1088.74	1021.97
Test Img-5	288.58	245.37	258.82	250.34	241.74
Test Img-6	508.00	435.16	475.08	430.37	420.58
Test Img-7	254.85	215.49	247.38	220.76	219.50
Test Img-8	265.08	215.33	249.65	236.48	222.85
Test Img-9	350.63	290.64	317.11	300.12	284.59
Test Img-10	212.66	174.86	192.98	184.07	173.04
Test Img-11	215.40	173.16	195.08	182.11	170.34
Test Img-12	250.54	196.85	234.35	205.05	201.49

3.3. Evolutionary Triangular Irregular Networks Refinement

Evolutionary Triangular Irregular Networks (TIN) are numerical tools used to model the *surface morphology* in Geomatics. TINs obtained by triangulating scattered points are frequently used to express topographical surfaces in Geographical Information Systems. TIN refinement is used in various computer graphic applications (Chernikov, & Chrisochoides , 2012). There are two commonly used methods for the refinement of the surfaces that are composed of triangular or gridded patches. The first method is based on reducing the number of vertex points or edges that are nused to create the digital mesh model of related surface. This method simplify the surface in accordance to a predefined error threshold value. The second method is based on iteratively updating of initial-triangulation by adding a new location-optimized vertex point to the existing triangular mesh. Initial triangulation includes only the outer boundary vertex points of the original mesh model. In this method, related iterative process is terminated when the vertex number of the triangular mesh reaches to the predetermined number of vertex. Hence, this method allows the identification of the surface with a predetermined number of vertex points. In this paper, the second method based TIN evolution process was employed for the refinement of test TIN model. The test TIN model were obtained by measuring the peak section of Mount Erciyes with a fixed-wing Atlasus UAV (Atlasus , 2018). The related measurements were made in geodesic datum of EPSG:5256 TUREF/TM36 (Turef , 2018). Mount Erciyes, which is an advanced ski resort, is one of the most beautiful mountains in the world and it is 3916m high. The experiments carried out to simplify the test TIN model, which represent the peak section of Mount Erciyes, were performed using ABC, JADE, CUCKOO, WDE, and BSD.

A step-by-step description of the evolutionary TIN refinement problem is given below as in (Bergen & Ross , 2012):

1. Create initial-TIN by using the only corner-vertex points of the Original-Mesh model.
2. Set the maximum number of vertex desired to be obtained at the end of the TIN evolution process.
3. Insert a new *location-optimized vertex* point to the current TIN and update triangulation.
4. Repeat Step 3 until reaching the desired number of vertex in the current TIN.

The computation phase of *location-optimized vertex* has been defined by using Eq. 18

$$\boxed{\begin{array}{l}\textbf{while}\; size\,(q) \le (size\,(q)+500)\\ \;\; \varepsilon_0 = callMeshObj\,(q, P_{Mesh})\\ \;\;\; \underset{s_{TIN}}{argmin} \quad \varepsilon = callMeshObj\left(\left[\begin{array}{cc} q & s_{TIN}\end{array}\right], P_{Mesh}\right)\\ \;\; If\;\; \varepsilon \prec \varepsilon_0 \quad then \quad q := \left[\begin{array}{cc} q & s_{TIN}\end{array}\right]\\ \textbf{endwhile}\end{array}} \tag{18}$$

where s_{TIN} is a new vertex inside the initial-TIN model. In Eq. 18, *callMeshObj* denotes objective function for TIN refinement and it has been defined by using Eq. 19.

$$\boxed{\begin{array}{l}\textbf{function} \quad \varepsilon = callMeshObj\,(q, P_{Mesh})\\ \;\left[\begin{array}{ccc} x_{Mesh} & y_{Mesh} & z_{Mesh}\end{array}\right] \leftarrow P_{Mesh}\\ \;\Delta_{Mesh} = Delaunay\,(x_{Mesh}, y_{Mesh})\\ \;[x_q, y_q] \leftarrow q\\ \;\Delta_q = Delaunay\,(x_q, y_q)\\ \;z_q = TLI\,(\Delta_{Mesh}, x_q, y_q)\\ \;z^*_{Mesh} = TLI\,(\Delta_q, x_{Mesh}, y_{Mesh})\\ \;\varepsilon = \sum\left(|z^*_{Mesh} - z_{Mesh}|\right)\end{array}} \tag{19}$$

where, *Delaunay* function, *TLI* denotes Delaunay triangulation process as in (Besdok, Civicioglu, & Alci , 2004; Civicioglu, & Alci , 2004) and Triangular Linear Interpolation (Besdok, Civicioglu, & Alci , 2004; Civicioglu, & Alci , 2004), respectively.

The *maximum number of vertex*=500 has been used in the experiments. The results obtained for the solution of the TIN refinement problem are illustrated in Fig.s 10-14. The points indicated in red in Fig.s 10-14 are the added vertex points to the initial-TIN model to converge original-TIN. In the solution of the TIN refinement problem, *size of pattern matrix* is set to 5, and *the number of function evaluation value* is set to 100. 30 different experiments were performed with each algorithm to solve the related TIN refinement problem. A different starting *pattern matrix* was used in each experiment. All algorithms used the same initial *pattern matrix* during the experiments.

Boxplot analysis of the results obtained in the TIN refinement experiments is shown in Fig 15.

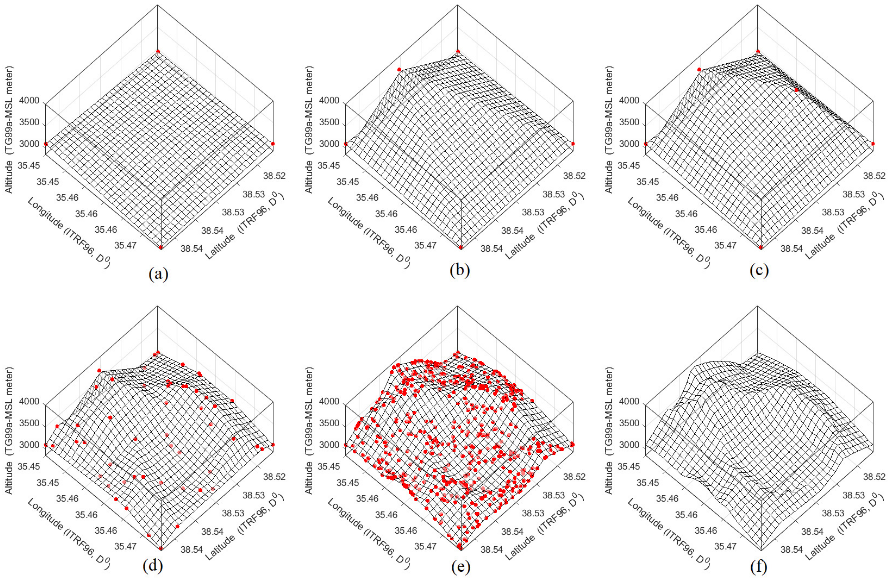


Figure 10. Evolution steps of TIN refinement problem by using ABC ; (a) Initial mesh, (b) Interpolated mesh after 1-point insertion into TIN, (c) Interpolated mesh after 2-points insertion into TIN, (d) Interpolated mesh after 50-points insertion into TIN, (e) Interpolated mesh after 500-points insertion into TIN, (f) Target-mesh model.

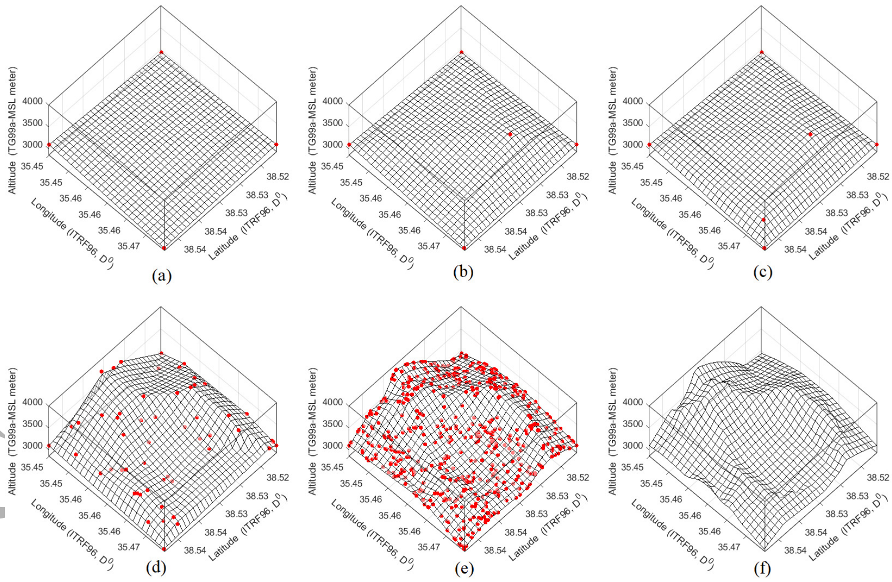


Figure 11. Evolution steps of TIN refinement problem by using JADE ; (a) Initial mesh, (b) Interpolated mesh after 1-point insertion into TIN, (c) Interpolated mesh after 2-points insertion into TIN, (d) Interpolated mesh after 50-points insertion into TIN, (e) Interpolated mesh after 500-points insertion into TIN, (f) Target-mesh model.

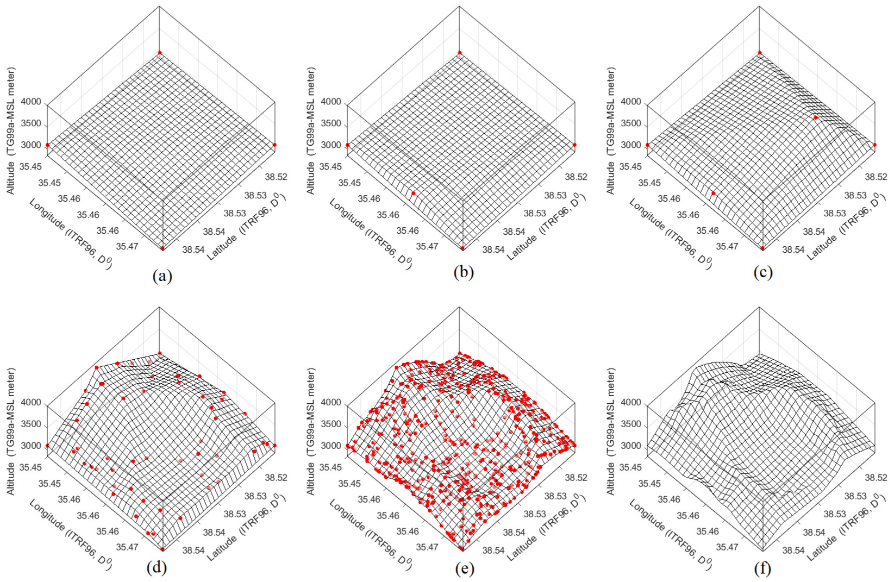


Figure 12. Evolution steps of TIN refinement problem by using CUCKOO ; (a) Initial mesh, (b) Interpolated mesh after 1-point insertion into TIN, (c) Interpolated mesh after 2-points insertion into TIN, (d) Interpolated mesh after 50-points insertion into TIN, (e) Interpolated mesh after 500-points insertion into TIN, (f) Target-mesh model.

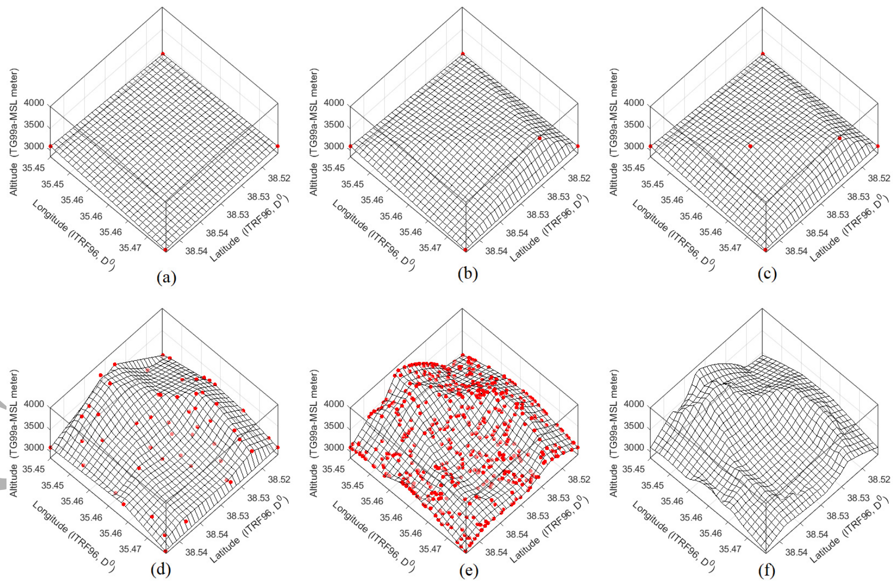


Figure 13. Evolution steps of TIN refinement problem by using WDE ; (a) Initial mesh, (b) Interpolated mesh after 1-point insertion into TIN, (c) Interpolated mesh after 2-points insertion into TIN, (d) Interpolated mesh after 50-points insertion into TIN, (e) Interpolated mesh after 500-points insertion into TIN, (f) Target-mesh model.

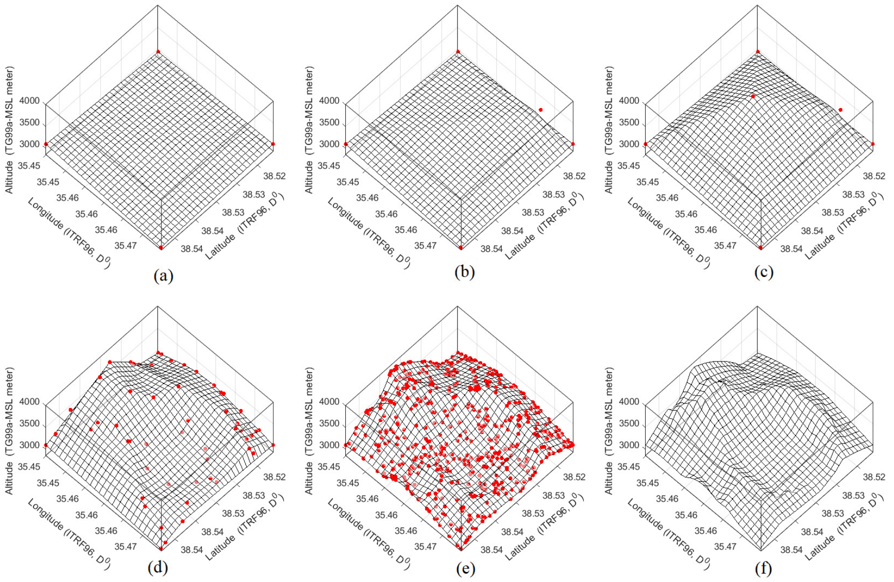


Figure 14. Evolution steps of TIN refinement problem by using BSD ; (a) Initial mesh, (b) Interpolated mesh after 1-point insertion into TIN, (c) Interpolated mesh after 2-points insertion into TIN, (d) Interpolated mesh after 50-points insertion into TIN, (e) Interpolated mesh after 500-points insertion into TIN, (f) Target-mesh model.

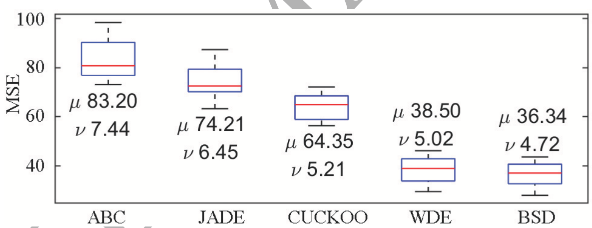


Figure 15. Boxplot analysis of TIN refinement results.

When the results given in Fig. 15 are analyzed, it is observed that BSD and WDE produced statistically very close results in the solution of the TIN refinement problem.

4. Conclusions

In evolutionary computation, it is difficult to determine the efficient *evolution direction* and *evolution step size* values. Determining the efficient evolutionary direction requires the use of global search strategies that can avoid local solutions. BSD uses span *pattern vector*s and the best solution available to produce efficient evolutionary direction vectors. Therefore BSD is a partially *elitist* algorithm. The evolutionary step size value controls the amplitude of direction vector. Efficient evolutionary step size is also very difficult to determine. The EA's search success can be sensitive to the nature of the random number generator used to generate related evolutionary step size value. BSD can use different types of random number generators to generate evolutionary step size value. BSD can scale each relevant parameter individually while solving problems involving *stongly related* or *highly correlated* parameters. Therefore, BSD can avoid local solutions while solving complex problems. As every *pattern vector* in BSD evolves towards a different *pattern vector*, BSD is a structurally bijective algorithm. The evolution of each *pattern vector* in BSD is independent of the evolution of other *pattern vectors*. This provides the *recursive* and *parallel* nature of BSD.

The statistical results obtained from the experiments show that BSD is capable of solving different types of digital problems and the problem solving success of BSD is highly better than the tested methods used in this paper.

The theoretical contributions of BSD are listed below:

1. Since the internal parameter values of the BSD are determined randomly, the BSD is practically a universal DE like WDE.
2. The mutation operator of BSD is structurally different from the mutation operator of ABC, JADE, CUCKOO and WDE.
3. BSD does not have mutation and crossover rate parameters.
4. BSD is a partially elitist method.
5. The crossover operator of the BSD is controlled using bezier polynomials. BSD's crossover operator is different from ABC, JADE, CUCKOO and WDE's crossover operators.
6. The structure of the BSD is very simple compared to the structures of the test methods.
7. The computational complexity of BSD is generally better than the computational complexity of ABC, JADE, CUCKOO and WDE.
8. BSD can work with very small and very large sized pattern matrix.
9. The ability of BSD to solve numerical problems is statistically better than those of the test methods.
10. Since BSD is a non-recursive method, solution vectors in BSD are evolved separately. Therefore, the operation of the BSD complies with parallel computing requirements.

Acknowledgements

Several sections of this paper have been supported by the following projects; Erciyes University BAP FDA-2013-4530, FBA-10-3067, FBA-9-1131, FBA-2013-4525, 06-AY-15 and Tubitak 115Y235.

References

Atlasus, http://atlasus.com.tr/Atlas/UAV (Access 18.12.2018)

Azhari, F., Heidarpour, A., & Zhao, X.L. (2018). On the use of Bernstain-Bezier functions for modelling the post-fire stress-strain relationship of ultra-high strength steel (Grade 1200). *Engineering Structures.* 175, 605-616.

Bergen, S., & Ross, J.R. (2009). Automatic and interactive evolution of vector graphics images with genetic algorithms. *The Visual Computer*, 28, 35-45.

Besdok, E., Civicioglu, P., & Alcı, M. (2004). Impulsive noise suppression from highly corrupted images by using resilient neural networks. *Lecture Notes in Artificial Intelligence*, 3070, 670-675.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer V. (2006). Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10, 646-657.

Chen, J., Zheng, J., Wu, P., et al. (2017). Dynamic particle swarm optimizer with escaping prey for solving constrained non-convex and piecewise optimization problems. *Expert Systems with Applications*, 86, 208-223.

Chernikov, A.N. & Chrisochoides, N.P. (2012). Generalized Insertion Region Guides For Delaunay Mesh Refinement, *SIAM Journal On Scientific Computing*, 34, A1333-A1350.

Civicioglu, P. (2013,a). Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219, 8121-8144.

Civicioglu, P. (2012). Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. *Computers & Geosciences*, 46, 229-247.

Civicioglu, P. (2013,b). Artificial cooperative search algorithm for numerical optimization problems. *Information Sciences*, 229 - 58-76.

Civicioglu, P., & Alci M. (2004). Impulsive noise suppression from highly distorted images with triangular interpolants. *AEU - International Journal of Electronics and Communications*, 58, 311-318.

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

Civicioglu, P., & Beşdok E. (2014). Comparative Analysis of the Cuckoo Search Algorithm, Cuckoo Search and Firefly Algorithm Theory and Applications, Springer, London, 85-113.

Civicioglu, P., Besdok, E., Gunen, M.A., & Atasever, U.H. (2018). Weighted differential evolution algorithm for numerical function optimization: a comparative study with cuckoo search, artificial bee colony, adaptive differential evolution, and backtracking search optimization algorithms. *Neural Computing and Applications.* Article online. https://doi.org/10.1007/s00521-018-3822-5 (Access 18.12.2018)

The matlab codes of classic benchmark problems (2019). *https : //www.mathworks.com/matlabcentral/fileexchange/*69827−*bernstain − search − differential − evolution − algorithm?s_tid=FX_rc2_behav* (last access 23.06.2019)

The matlab codes of classic benchmark problems (2019). *https : //www.sfu.ca/ ssurjano/optimization.html* (last access 23.06.2019)

Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6, 58-73.

Das, S., Mullick, S.S., & Suganthan, P.N. (2016). Recent advances in differential evolution - An updated survey. *Swarm and Evolutionary Computation*, 27, 1-30.

Derrac, J., Garca, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1 3-18.

Karaboğa, D., & Basturk B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39, 459-471.

Liang, J.J., & Qu, B.Y., & Suganthan, P.N. (2013). Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Technical Report 201311, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore, December 2013.

Liu, J., & Zhang, H., & He, K. & et al. (2018). Multi-objective particle swarm optimization algorithm based on objective space division for the unequal-area facility layout problem. *Expert Systems with Applications*, 102, 179-192.

Lynn, N., & Suganthan P.N., (2017).Ensemble particle swarm optimizer. *Applied Soft Computing*, 55, 533-548.

Opara K,Arabas J (2018). Comparison of mutation strategies in Differential Evolution – A probabilistic perspective, *Swarm and Evolutionary Computation.* 39, 53–69.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8, 3-30.

Mathworks, Matlab File Exchange, (2019), https://www.mathworks.com/matlabcentral/fileexchange/69819-bernstain-search-differential-evolution-algorithm (Last access 25.06.2019)

Mlakar, U., Potocnik, B., & Brest, J., (2016). A hybrid differential evolution for optimal multilevel image thresholding. *Expert Systems with Applications*, 65, 221-232.

Mohamed, A.W., & Suganthan P.N., (2018). Real-parameter unconstrained optimization based on enhanced fitness-adaptive differential evolution algorithm with novel mutation . *Soft Computing*, 22, 3215-3235.

Price, K.V., Storn, R., & Lampinen, J. (2005). Differential evolution: A practical approach to global optimization. Springer, Berlin, Germany.

Özsoydan, F.B., & Baykaşoğlu, A., (2019). Quantum firefly swarms for multimodal dynamic optimization problems. *Expert Systems with Applications*, 115, 189-199.

Qin, Q., Cheng, S., Zhang, Q., & et al. (2014). Multiple strategies based orthogonal design particle swarm optimizer for numerical optimization. *Computers & Operations Research*, 60, 91-110.

Zhang, Q., Zou, D., Duan, N., et al. (2019). An adaptive differential evolutionary algorithm incorporating multiple mutation strategies for the economic load dispatch problem. *Applied Soft Computing*, 78, 641-669.

Turef, https://epsg.io/5256 (Last access 25.06.2019)

Yang, X.S., & Deb, S. (2009). Cuckoo search via levy flights. World Congress on Nature and Biologically Inspired Computing-Nabic'2009, Coimbatore, India, 4, 210-214.

Zhang, J., & Sanderson, A.C. (2009). JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13, 945-958.

Zhang, W.B., & Zhu, G.Y. (2011). Comparison and application of four versions of particle swarm optimization algorithms in the sequence optimization. *Expert Systems with Applications*,Volume: 38 Issue: 7 Pages: 8858-8864 Published: JUL 2011

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

AUTHORSHIP STATEMENT

Manuscript title: ____

Bernstain-Search Differential Evolution Algorithm for Numerical Function Optimization

____

____

All persons who meet authorship criteria are listed as authors, and all authors certify that they have participated sufficiently in the work to take public responsibility for the content, including participation in the concept, design, analysis, writing, or revision of the manuscript. Furthermore, each author certifies that this material or similar material has not been and will not be submitted to or published in any other publication before its appearance in the Expert Systems with Applications. .

Authorship contributions

Please indicate the specific contributions made by each author (list the authors' initials followed by their surnames, e.g., Y.L. Cheung). The name of each author must appear at least once in each of the three categories below.

Category 1

Civicioglu, P., Besdok, E.

acquisition of data: Civicioglu, P., Besdok, E.

analysis and/or interpretation of data: Civicioglu, P., Besdok, E.

Category 2

Drafting the manuscript: Civicioglu, P., Besdok, E.

revising the manuscript critically for important intellectual content: Civicioglu, P., Besdok, E.

____________, ____________.

Category 3

Approval of the version of the manuscript to be published (the names of all authors must be listed):

Civicioglu, P., Besdok, E.

______________, ______________, ______________, ______________, ______________,

______________, ______________, ______________, ______________, ______________.

Acknowledgements

All persons who have made substantial contributions to the work reported in the manuscript (e.g., technical help, writing and editing assistance, general support), but who do not meet the criteria for authorship, are named in the Acknowledgements and have given us their written permission to be named. If we have not included an Acknowledgements, then that indicates that we have not received substantial contributions from non-authors.

This statement is signed by all the authors (*a photocopy of this form may be used if there are more than 10 authors***):**

Author's name (typed)	Author's signature	Date
Pinar Civicioglu		26.06.2019
Erkan Besdok		26.06.2019