Available online at www.sciencedirect.com

ScienceDirect

Expert Systems with Applications 00 (2019) 1–6

Bernstain-Search Differential Evolution Algorithm : BSD

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

1. Cite As ;

Civicioglu, P., Besdok, E., (2019), Bernstain-search differential evolution algorithm for numerical function optimization, Expert Systems with Applications, 138 (30), 112831

Terminology ;
"Bernstain polynomials" (Azhari & et al , 2018) == "Bernstein polynomials" (Badea, , 2011)

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 vector*s are calculated using Eq 2;

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

*Corresponding author

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

[1]Tel.: +xxx-xx fax: +xxx-xx.

[2]Tel.: +xxx-xx; fax: +xxx-xxx.

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;

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

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 $\kappa_0 = \lceil 3 \cdot \kappa_1^3 \rceil, \kappa_1 \sim \mathbf{U}[0\ 1], \ \ \kappa_0 \in \mathbf{U}\{1:3\}$. 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;

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

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.

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

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

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

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 vector*s 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.

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

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.

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 $\kappa_0 = \lceil 3 \cdot {\kappa_1}^3 \rceil$, $\kappa_1 \sim U[0\ 1]$, $\kappa_0 \in \mathbf{U}\{1:3\}$
    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 Bernstain-Search Differential Evolution Algorithm (BSD). The unoptimized Matlab code of the BSD is publicly available at (Mathworks , 2019).

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 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;

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

Here $s = 0 : n$, $\begin{pmatrix} n \\ s \end{pmatrix} = \frac{n!}{s!(n-s)!}$. 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 (Azhari & et al , 2018) polynomials for $0 \leq t \leq 1$;

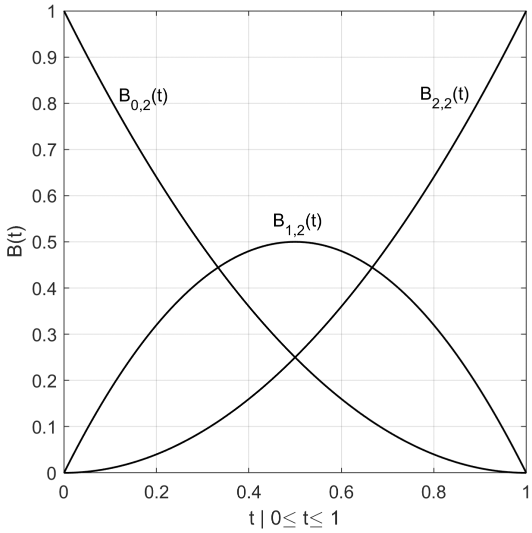

Figure 2. 2^{nd} degree Bernstain polynomials.

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

Badea, C. Bernstein Polynomials and Operator Theory. *Results. Math.* (2009) 53: 229. https://doi.org/10.1007/s00025-008-0333-1