ISSN: 1803-3814 (Printed), 2571-3701 (Online)
https://doi.org/10.13164/mendel.2019.1.065

MENDEL Soft Computing Journal

# EIGENVECTOR CROSSOVER IN THE EFFICIENT jSO ALGORITHM

**Petr Bujok[1,✉], Radka Poláková[2]**

[1]Department Informatics and Computers, University of Ostrava, Czech Republic
[2]Department of Social Sciences, University of Ostrava, Czech Republic
petr.bujok@osu.cz[✉], radka.polakova@osu.cz

**Abstract**

*In this paper, a new variant of an efficient adaptive jSO algorithm is presented. The original jSO uses popular binomial crossover which is applied in a standard coordinate system. Many problems tend to rotate the coordinate system in one or more axes. This is the reason why a crossover variant using Eigen coordinate system replaces the original binomial version of crossover in jSO. The newly proposed jSOe performs significantly better compared with the original jSO when solving 90 problems of the CEC 2017 benchmark set.*

**Keywords:** *differential evolution, eigenvector crossover, jSO, experimental comparison, CEC 2017.*

**Received:** 4 April 2019
**Accepted:** 30 May 2019
**Published:** 24 June 2019

## 1 Introduction

Global optimisation research tampers with many fields of science, business, industry, or healthcare, where optimal settings are required. In this paper, the objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, x_2, \ldots, x_D) \in \mathbb{R}^D$ defined on the search domain $\Omega$ limited by lower and upper boundaries, i.e. $\Omega = \prod_{j=1}^{D}[a_j,\ b_j],\ a_j < b_j,\ j = 1, 2, \ldots, D$, is considered. The global minimum point $\boldsymbol{x}^*$, which satisfies condition $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \Omega$, is the required solution with the minimal function value.

There are a lot of optimisation techniques to localise the solution of problems. One group of methods is widely used at the expense of other ones, the group is called Evolutionary algorithms (EA). In 1995, a very popular EA called Differential evolution (DE) algorithm was introduced by Storn and Price [10, 11]. Differential evolution is a very simple optimiser using the population of individuals to search for the solution of the task. Although DE performs well, there are a lot of problems that DE is not able to solve. During the last decades, many efficient adaptive variants of DE have been proposed and applied to various optimisation problems [5, 6].

The main idea of this paper is to apply a new type of crossover in the efficient adaptive jSO algorithm. This approach promises better results especially in problems represented by rotated functions. The rest of the paper is organised as follows. A short description of jSO and some applications are presented in Section 2. Several applications of Eigenvector crossover in DE and the newly proposed algorithm are described in Section 3. The setting of the experiment and related results are presented and discussed in Section 4 and 5. The last section concludes the paper.

## 2 Adaptive jSO Variant

One of the most efficient DE proposed in 2017 is called *jSO* [3]. This algorithm was the best performing DE algorithm in Congress on Evolutionary Computation (CEC) 2017, and it took overall the second place. The jSO algorithm was derived step by step from successful JADE, SHADE, L-SHADE, and iL-SHADE algorithms. More details about jSO control parameters and settings are provided in [3].

The jSO algorithm introduces a weighted version of the popular current-to-$p$best mutation strategy (current-to-$p$best$_w$). In addition, an archive $A$ to store outperformed parent individuals is used. Circle memories to adapt the main DE control parameters $F$ and $CR$ are inherited from the preceding SHADE and L-SHADE versions, where $\mu_{\mathrm{CR}} = 0.8$ and $\mu_F = 0.5$. In jSO, the same initial values for $\mu_{\mathrm{CR}}$ are used whereas smaller values of $F$ are preferred for $\mu_F$, $\mu_F = 0.3$. Moreover, both mean values on the last $H$th positions are set to the same value, $\mu_{\mathrm{CR}} = \mu_F = 0.9$. A linear reduction of the population size is used in L-SHADE, too. The main points in which jSO differs from the preceding variants are described in the original paper [3] in more detail. The values of jSO parameters are self-adapted during the search process, and if one uses their recommended initial values, this method is parameter-free.

The jSO algorithm is a very efficient algorithm indirectly derived from the JADE algorithm. Piotrowski et al. studied more than 20 variants of the JADE algorithm on two sets of problems (CEC 2014 and CEC 2011) [8]. The results provide information about the robustness of JADE-based methods, artificial benchmark problems and also real-world problems were used.

The results of the population-size control mechanism [9] show that adaptation by the diversity-based mechanism is beneficial for many DE variants except for jSO. The reason is probably that jSO has very sophisticated adaptive settings and it is not easy to increase its efficiency.

Although the jSO algorithm has a fine-tuned setting of the control parameters and no more increase is obvious, a model of eight cooperating jSO algorithms in a parallel topology provides good performance in some the real-world problems [4].

## 3 Eigenvector Crossover in DE

The newly proposed jSO variant is constructed to increase the efficiency of original jSO. The original binomial crossover is replaced in jSO by new Eigenvector crossover. In 2014, a new approach to cope with rotated objective functions was introduced in the CoBiDE algorithm [12]. The covariance-based Eigenvector coordinate system for a crossover operation was studied in CoBiDE. The main idea was to tackle problems with correlated coordinates. The idea of this mechanism is very simple as described in following. In 2015, Guo et al. introduced SPS-L-SHADE-EIG algorithm [7] for competition CEC 2015. This algorithm was the winner of the competition. Guo uses the same transformation mechanism as was used in CoBiDE. In SPS-L-SHADE-EIG, for each individual, a decision is made, to use standard binomial or new Eigenvector crossover. For each individual, a probability of Eigenvector crossover is generated from Gauss distribution with the mean value located in a circle memory (similarly for parameters $F$ and $CR$). Similarly, Awad et al. in 2018 introduce the L-covnSHADE algorithm with Covariance-based crossover [2]. New individuals are developed using a standard binomial crossover and Eigenvector crossover is used for points selected from Euclidean neighbourhood of the best point of population. The approach is applied to a set of CEC 2011 real-world problems. The results show that the proposed method is at least comparable with the original L-SHADE and four adaptive DE.

### 3.1 Eigenvector Crossover for jSO

The Eigenvector crossover used in this experiment was introduced in CoBiDE [12]. At first, a covariance matrix $C$ from a certain part of the population (controlled by input parameter $ps \in (0,\ 1)$) is computed. The higher the value of $ps$, the bigger the part of the population is selected. Matrix $C$ is decomposed into two matrices containing Eigenvectors $B$ and Eigenvalues $D$:

$$C = BD^2B^T. \tag{1}$$

In each generation, the original binomial crossover or the new Eigenvector crossover is applied to all points of the population. The selection of the Eigenvector crossover is controlled by probability equal to input parameter $pb \in (0,\ 1)$. The higher the value of $pb$, the higher the probability of using the Eigenvector crossover.

When, the Eigenvector crossover is selected, each mutated point $\boldsymbol{u}_i$ is combined with the parent point $\boldsymbol{x}_i$ in the Eigen coordinate system:

$$\boldsymbol{x}_i^{'} = B^T\boldsymbol{x}_i \qquad \text{and} \qquad \boldsymbol{u}_i^{'} = B^T\boldsymbol{u}_i. \tag{2}$$

Then the original binomial crossover is applied to the points in the new coordinate system $(\boldsymbol{x}_i^{'}, \boldsymbol{u}_i^{'})$. After crossover, the new trial point $\boldsymbol{y}_i^{'}$ is transformed from the Eigen coordinate system back to the original coordinate system.

$$\boldsymbol{y}_i = B\boldsymbol{y}_i^{'} \tag{3}$$

### 3.2 Proposed jSOe

It was mentioned that in this paper, the original binomial crossover in jSO is replaced by the Eigenvector crossover adopted from the CoBiDE variant. The steps of a new *jSOe* variant are shown in Algorithm 1. The pseudo-code describes the original jSO algorithm, the new approach differs only in highlighted rows.

At first, the population of $N$ potential solutions is initialised and evaluated by the objective function. Also, circle memories for $F$ and $CR$ parameters, and two control parameters for the Eigenvector crossover are initialised. Before each generation of the search, an uniformly distributed random number is compared with the first control parameter $pb$. If $rand < pb$, all individuals in this generation are updated using the Eigenvector crossover, and vice versa.

For each point, mean values for generating $F$ and $CR$ parameters $M_F$ and $M_{\mathrm{CR}}$ are selected by a roulette wheel. The control parameters of jSO are generated by Gauss ($CR$) and Cauchy ($F$) distributions using the mean values. The authors of jSO recommended to truncate $F$ and $CR$ to certain values based on the current time of the search process. These settings make jSO a fine-tuned optimisation algorithm, and it seems that there is no more possibility to increase the performance of jSO.

**Algorithm 1** jSO algorithm with Eigenvector Crossover

archive $A \leftarrow \emptyset$
initialise $ps$, $pb$ ⇐
initialise population $P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
set all values of $M_F$ to 0.5
set all values of $M_{\mathrm{CR}}$ to 0.8
**while** stopping condition not reached **do**
  $S_{\mathrm{CR}} \leftarrow \emptyset$, $S_F \leftarrow \emptyset$
  **if** $rand < pb$ **then**
    $eig \leftarrow 1$ ⇐
  **else**
    $eig \leftarrow 0$ ⇐
  **end if**
  **for** $i = 1, 2, \ldots, N$ **do**
    $r_i \leftarrow$ select from $[1,\ H]$ randomly
    **if** $r_i = H$ **then**
      $M_{F,r_i} \leftarrow 0.9$
      $M_{\mathrm{CR},r_i} \leftarrow 0.9$
    **end if**
    **if** $M_{\mathrm{CR},r_i} < 0$ **then**
      $CR_i \leftarrow 0$
    **else**
      $CR_i \leftarrow N_i(M_{\mathrm{CR},r_i},\ 0.1)$
    **end if**
    **if** $g < 0.25 G_{\max}$ **then**
      $CR_i \leftarrow max(CR_i,\ 0.7)$
    **else if** $g < 0.5 G_{\max}$ **then**
      $CR_i \leftarrow max(CR_i,\ 0.6)$
    **end if**
    $F_i \leftarrow C_i(M_{F,r_i},\ 0.1)$
    **if** $g < 0.6 G_{\max}$ and $F_i > 0.7$ **then**
      $F_i \leftarrow 0.7$
    **end if**
    **if** $eig = 1$ **then**
      $\boldsymbol{x}_i,\ \boldsymbol{u}_i \rightarrow \boldsymbol{x}_i^{'},\ \boldsymbol{u}_i^{'}$ (2) ⇐
    **end if**
    $\boldsymbol{y}_i \leftarrow$ current-to-$p$best$_w$/1/bin
    **if** $eig = 1$ **then**
      $\boldsymbol{x}_i^{'},\ \boldsymbol{u}_i^{'} \rightarrow \boldsymbol{x}_i,\ \boldsymbol{u}_i$ (3) ⇐
    **end if**
    compute $f(\boldsymbol{y}_i)$
  **end for**
  **for** $i = 1, 2, \ldots, N$ **do**
    **if** $f(\boldsymbol{y}_i) \leq f(\boldsymbol{x}_i)$ **then**
      $\boldsymbol{x}_i \leftarrow \boldsymbol{y}_i$
    **end if**
    **if** $f(\boldsymbol{y}_i) < f(\boldsymbol{x}_i)$ **then**
      $\boldsymbol{x}_i \rightarrow A$, $CR_i \rightarrow S_{\mathrm{CR}}$, $F_i \rightarrow S_F$
    **end if**
    update $M_{\mathrm{CR}}$ and $M_F$
    update population size
    update $p$
  **end for**
**end while**

Having set the control parameters, a mutation point is generated using a novel-mutation variant (4)

$$\boldsymbol{u}_i = \boldsymbol{x}_i + F_w(\boldsymbol{x}_{\text{pBest}} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}), \tag{4}$$

where $\boldsymbol{x}_i$ is the current point, $\boldsymbol{x}_{\text{pBest}}$ is randomly selected point from $p \times N$ the best points of $P$, $\boldsymbol{x}_{r_1}$ is selected randomly from $P$, and $\boldsymbol{x}_{r_2}$ is selected randomly from $P \cup A$. The only change is in newly used parameter $F_w$, computed using (5):

$$F_w = \begin{cases} 0.7 \times F, & FES < 0.2 \times maxFES \\ 0.8 \times F, & FES < 0.4 \times maxFES \\ 1.2 \times F, & otherwise. \end{cases} \tag{5}$$

The portion of the best individuals to select $\boldsymbol{x}_{\text{pBest}}$ point ($p$) is adapted during the search using the following formula (6):

$$p = \frac{p_{\max} - p_{\min}}{maxFES} \times FES + p_{\min}, \tag{6}$$

where $p_{\max}$, $p_{\min}$ are input parameters, $maxFES$ is the total number of function evaluation per run and $FES$ is the current number of function evaluations. The authors proposed to use $p_{\max} = 0.25$, and $p_{\min} = 0.125$.

After mutation, the current point $\boldsymbol{x}_i$ and the mutation point $\boldsymbol{u}_i$ are transformed to the Eigen coordinate system using (2) if the randomly selected number was less than $pb$. Then, the binomial crossover is applied to generate a new trial point $\boldsymbol{y}_i'$. This point is located in the Eigen coordinate system and therefore it is transformed back in the original coordinate system by formula (3). After evaluation, a new point replaces the current $\boldsymbol{x}_i$ only if $f(\boldsymbol{y}_i) \leq f(\boldsymbol{x}_i)$. The generation being completed, the parameters of jSOe are updated.

## 4 Experiments

The test suite of 30 problems was proposed for a special session and competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization, a part of Congress on Evolutionary Computation (CEC) 2017. This session was intended as a competition of optimisation algorithms where new variants of algorithms are introduced. The functions are described in [1], including the experimental settings required for the competition. The source code of the functions is also available on the web site given in the report [1]. The test functions CEC 2017 are divided into four categories, based on their difficulty:

- unimodal functions - simple problems: $F1 - F3$,
- multimodal functions - with many local minima: $F4 - F10$,
- hybrid functions - difficult, considered as the real-world problems: $F11 - F20$,
- composition functions - very difficult, composed of several different functions: $F21 - F30$.

The *jSOe* algorithm is implemented in Matlab 2017b and this environment was also used for the experiments. All computations were carried out on a standard PC with Windows 7, Intel(R) Core(TM)i7-4790 CPU 3.6 GHz, 16 GB RAM. The experimental setting follows the requirements given in report [1], where 30 minimisation problems are also defined. The source code of the functions in C was downloaded from the web page given in [1] and compiled by Lcc-win32 C 2.4.1 compiler. The search range (domain) for all the test functions is $[-100, 100]^D$.

Our tests were carried out at three levels of dimension, $D = 10$, 30, 50, with 51 independent runs per each test function. Each point in the population is evaluated by the cost function. The function-error value is computed as the difference between the function value of the current point with a minimal function value and the known function value in the global minimum point of each test problem. The run of the algorithm stops if the prescribed amount of function evaluation $MaxFES = D \times 10^4$ is reached or if the minimum function error in the population is less than $1 \times 10^{-8}$. Such an error value is considered sufficient for an acceptable approximation of the correct solution. The values of the function error less than $1 \times 10^{-8}$ are treated as zero in further processing. The population size of all algorithms in the comparison are initialised at the same value $N_{\text{init}} = 25 \times \log(D) \times \sqrt{D}$. The control parameters of the Eigenvector crossover are studied. Beside the original settings from CoBiDE ($ps = 0.5$, $pb = 0.4$), four different combinations are used, where the 0.1 and 0.9 values are applied. Abbreviations of five variants of jSOe are '*jSOe*'+'*ps*'+'*pb*' (i.e. jSOe0504).

## 5 Results

Five variants of the newly proposed jSOe algorithm are compared with the original jSO using 90 test problems (including three dimensions). A brief insight into the performance of all six algorithms is provided by the Friedman test. This method computes the average ranks of the algorithms using the medians of minimal function value errors in each dimension. The mean ranks for each algorithm and dimension are depicted in Figure 1, and the results for each algorithm are linked by a solid line. The null hypotheses on equivalent efficiency of the algorithms were rejected for all the levels of dimension with an achieved significance level $p < 5 \times 10^{-5}$. The results show how the performance varied with increasing dimension. The most dramatic change is observed for jSOe0509 and jSOe0904. The first variant prefers the Eigenvector crossover and its performance for $D = 30$ is the worst in the comparison, but for other dimensions, it outperforms the original jSO at least. Including results of all three dimensions, the best-performing algorithms are jSOe0504 and jSOe0501 (both variants achieve the same average mean rank). The worst average mean rank is observed for the original jSO. These brief results provided the following facts. The Eigenvector crossover increases the efficiency of the jSO algorithm. The efficiency of the jSOe variant is higher if the middle setting of $ps = 0.5$ is used.

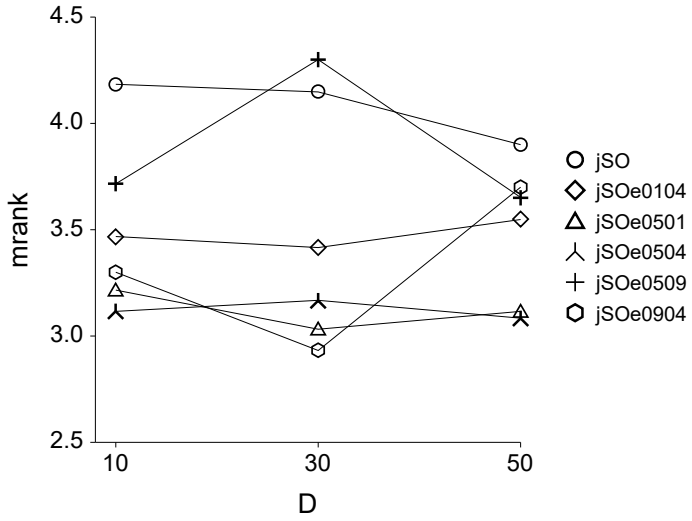


Figure 1: Mean ranks of all jSO variants from the Friedman test.

Table 1: Best, second best, and worst positions by the Kruskal-Wallis and Dunn's tests.

| Pos | D10 | jSO | $jSOe_{0104}$ | $jSOe_{0904}$ | $jSOe_{0501}$ | $jSOe_{0509}$ | $jSOe_{0504}$ |
|---|---|---|---|---|---|---|---|
| best | 10 | 9 | 3 | 7 | 4 | 5 | 2 |
| best | 30 | 7 | 5 | 4 | 12 | 1 | 1 |
| best | 50 | 7 | 5 | 4 | 12 | 1 | 1 |
| 2nd | 10 | 0 | 8 | 3 | 5 | 4 | 10 |
| 2nd | 30 | 1 | 5 | 9 | 2 | 5 | 8 |
| 2nd | 50 | 1 | 5 | 9 | 2 | 5 | 8 |
| last | 10 | 22 | 2 | 0 | 0 | 6 | 0 |
| last | 30 | 14 | 1 | 1 | 1 | 13 | 0 |
| last | 50 | 14 | 1 | 1 | 1 | 13 | 0 |

More details of the comparison is provided by the Kruskal-Wallis non-parametric one-way ANOVA test applied to each test problem. It was found that the performance of the algorithms in the comparison significantly differs. Dunn's method was then applied for a multiple comparison. The counts of the best positions, the second best positions, and the worst positions of the Dunn test are in Table 1. When studying the best positions, it seems that jSOe0504 is the worst performing method in the comparison, but it mostly takes the second position out of six algorithms. Moreover, this variant never achieves the worst results. When including all results, jSOe0501 achieves the best performance, because it is mostly on the first position, and very rarely on the last position. This variant uses the original setting of $ps = 0.5$ and applies the Eigenvector crossover with small probability, $pb = 0.1$. The original jSO algorithm is the worst performing method in most cases. On the other

hand, jSO also provides the best results in some problems. Regarding the performance of *ps* and *pb* parameters, smaller values of *pb* are preferred, and rather a higher portion of population *ps* achieves better results. It is necessary to note that in this table, there are not all 90 problems, because in some cases, the difference was not significant.

The performance of jSOe setting, i.e. *ps* and *pb* parameters, is assessed by the Wilcoxon rank-sum test. Each variant of jSOe is compared with the original jSO on each test function and dimension level. Table 2 contains counts of problems where the newly proposed variant or original jSO performs significantly better. It is obvious that each jSOe variant achieves better results in more problems than the original jSO. The worst performing jSOe variant, jSOe0509, performs better in 31 problems, and worse in 22 problems out of 90. The best results for $D = 10$ are provided by jSOe0501, for $D = 30$ jSOe0904 is the best performing and for $D = 50$ all jSOe variants perform similarly, except the worst jSOe0509. Based on the number of the wins and losses, the best performing algorithm is jSOe0501.

Table 2: Wins and losses provided by the Wilcoxon rank-sum test.

| $D$ | Alg. | $\text{jSOe}_{0104}$ | $\text{jSOe}_{0904}$ | $\text{jSOe}_{0501}$ | $\text{jSOe}_{0509}$ | $\text{jSOe}_{0504}$ |
|---|---|---|---|---|---|---|
| 10 | jSOe | 10 | 12 | 14 | 10 | 14 |
| 10 | jSO | 1 | 2 | 0 | 4 | 1 |
| 30 | jSOe | 13 | 16 | 15 | 9 | 15 |
| 30 | jSO | 2 | 1 | 1 | 8 | 2 |
| 50 | jSOe | 15 | 16 | 13 | 12 | 17 |
| 50 | jSO | 5 | 7 | 5 | 10 | 6 |
| $\Sigma$ | jSOe | 38 | 44 | 42 | 31 | 46 |
| $\Sigma$ | jSO | 8 | 10 | 6 | 22 | 9 |

More details are provided in Table 3, where the counts of significant wins and losses from the Wilcoxon rank-sum test are computed for four different types of problems (unimodal, multimodal, hybrid, and composition). Such a division provides details of the performance of the proposed jSOe algorithm. For the lower dimension and unimodal problems, all algorithms perform similarly, whereas, with increasing dimension, the performance of jSOe rises. For multimodal problems and $D > 10$, the situation is different. For $D = 30$, the worst performing jSOe (jSOe0509) performs worse than jSO, and for $D = 50$ all jSOe variants perform rather worse than the original jSO. In the case of hybrid problems (mentioned as real-world problems), all jSOe variants perform better than jSO, only the worst performing jSOe0905 achieves the same results for $D = 30$ as the original jSO. The most complex composition problems are very hard problems. Here, jSOe provides better results compared with the original jSO, only the worst performing jSOe0509 performs for $D = 50$ similarly to the jSO.

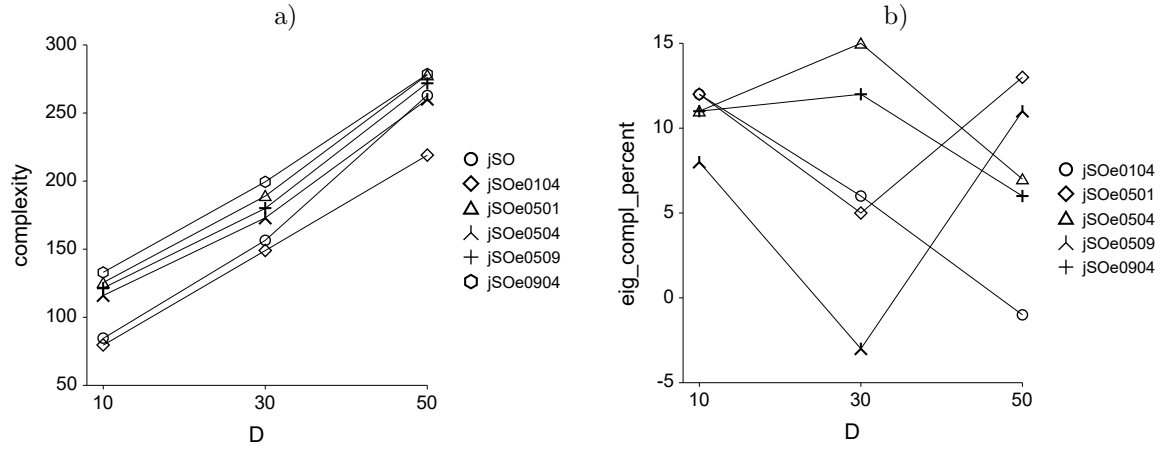


Figure 2: a) estimated time complexity of jSO variants; b) eigenvector and standard crossover complexity ratio in percent.

One can argue that although the new Eigenvector crossover increases the efficiency of jSO, it necessarily increases the complexity of jSO and related time demands. The time complexity of jSO and jSOe variants are estimated by solving $F2$ problem as it is described in report CEC 2017 [1]. The estimated time complexity of all algorithms and three dimensions is depicted in Figure 2 a). Estimates of the time complexity of each

Table 3: Wins of algorithms by the Wilcoxon rank-sum test.

| $D$ | Fun | Alg. | jSOe$_{0104}$ | jSOe$_{0904}$ | jSOe$_{0501}$ | jSOe$_{0509}$ | jSOe$_{0504}$ |
|---|---|---|---|---|---|---|---|
| 10 | uni | jSOe | 0 | 0 | 0 | 0 | 0 |
| 10 | uni | jSO | 0 | 0 | 0 | 0 | 0 |
| 10 | multi | jSOe | 3 | 2 | 5 | 2 | 4 |
| 10 | multi | jSO | 0 | 0 | 0 | 1 | 0 |
| 10 | hybrid | jSOe | 2 | 5 | 5 | 5 | 5 |
| 10 | hybrid | jSO | 1 | 2 | 0 | 3 | 1 |
| 10 | comp | jSOe | 5 | 5 | 4 | 3 | 5 |
| 10 | comp | jSO | 0 | 0 | 0 | 0 | 0 |
| 30 | uni | jSOe | 2 | 2 | 2 | 2 | 2 |
| 30 | uni | jSO | 0 | 0 | 0 | 0 | 0 |
| 30 | multi | jSOe | 2 | 2 | 2 | 0 | 2 |
| 30 | multi | jSO | 0 | 0 | 0 | 3 | 0 |
| 30 | hybrid | jSOe | 5 | 7 | 7 | 4 | 6 |
| 30 | hybrid | jSO | 1 | 1 | 1 | 4 | 1 |
| 30 | comp | jSOe | 4 | 5 | 4 | 3 | 5 |
| 30 | comp | jSO | 1 | 0 | 0 | 1 | 1 |
| 50 | uni | jSOe | 3 | 2 | 2 | 2 | 3 |
| 50 | uni | jSO | 0 | 0 | 0 | 0 | 0 |
| 50 | multi | jSOe | 2 | 2 | 2 | 1 | 2 |
| 50 | multi | jSO | 3 | 4 | 3 | 5 | 4 |
| 50 | hybrid | jSOe | 6 | 8 | 6 | 6 | 8 |
| 50 | hybrid | jSO | 1 | 1 | 1 | 2 | 1 |
| 50 | comp | jSOe | 4 | 4 | 3 | 3 | 4 |
| 50 | comp | jSO | 1 | 2 | 1 | 3 | 1 |
| $\Sigma$ | uni | jSOe | 5 | 4 | 4 | 4 | 5 |
| $\Sigma$ | uni | jSO | 0 | 0 | 0 | 0 | 0 |
| $\Sigma$ | multi | jSOe | 7 | 6 | 9 | 3 | 8 |
| $\Sigma$ | multi | jSO | 3 | 4 | 3 | 9 | 4 |
| $\Sigma$ | hybrid | jSOe | 13 | 20 | 18 | 15 | 19 |
| $\Sigma$ | hybrid | jSO | 3 | 4 | 2 | 9 | 3 |
| $\Sigma$ | comp | jSOe | 13 | 14 | 11 | 9 | 14 |
| $\Sigma$ | comp | jSO | 2 | 2 | 1 | 4 | 2 |

algorithm are linked by a solid line. It is obvious that the time complexity of all methods is increasing with increasing dimension. Moreover, when the value of the parameter $ps$ is increased, estimated time-complexity is also increased. It shows that using a larger portion of population for computing the covariance matrix causes significantly higher time-complexity. There is no obvious influence of the second parameter of the proposed method on jSO time-complexity. Figure 2 b) illustrates the time-complexity increase in percent when Eigenvector crossover is used instead of standard binomial crossover in jSOe variants. The positive value here means that jSOe variant have larger time-complexity than original jSO, negative value describes the situation when the jSOe variant is less time-complex than original jSO. The time are measured in generation and the average time of these values is used for drawing the Figure. The variants of jSO with Eigenvector crossover are rather more complex, but maximal level of ratio is 15 % for the original setting $ps = 0.5$ and $pb = 0.4$. Rather surprising is low time-complexity of Eigenvector crossover in variant jSOe0509 for $D = 30$, where this crossover is used in 90 % of generations. The experiments of complexity are repeated five times and are calibrated to 'neutral' CPU time. The time complexity of jSOe0104 is less than the time complexity of the original jSO. The time complexity of jSO is increased more than the time complexity of jSOe variants. These results were not expected; it is necessary to note that all experiments of time complexity were performed on the same PC.

## 6 Conclusion

In this paper, the Eigenvector crossover was applied to the successful jSO algorithm. Based on the two control parameters of the approach, five different variants of the newly proposed jSOe were developed. All six jSO algorithms are used on 90 test problems of CEC 2017. The results of the Friedman test show that the Eigenvector crossover increases the efficiency of the jSO algorithm. The efficiency of the jSOe variants is higher when the middle value of $ps = 0.5$ is used. More particular results of the Kruskal-Wallis test highlight the best performance of jSOe0501, because it is mostly in the first position, and very rarely in the last position. The original jSO algorithm is the worst performing method in most cases of the comparison. Smaller values of $pb$ are preferred and rather higher portion of population $ps$ achieves better results. A comparison of all jSOe variants with the jSO algorithm by the Wilcoxon rank-sum test shows that jSOe performs better in more problems than the original jSO algorithm. The same statistical test was applied to the results divided into four types of problems. All jSOe variants perform better than the jSO algorithm, the results are more balanced for multimodal problems, where jSO occasionally performs better. The estimated time complexity showed small computational demands of the applied Eigenvector crossover. The aforementioned points result in further research in the application of the Eigenvector crossover in advanced DE variants.

## References


[1] Awad, N. H., Ali, M. Z., Liang, Jing J., Qu, B. Y., and Suganthan, P. N. 2016. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization.* Nanyang Technological University, Singapore and Jordan University of Science and Technology, Jordan and Zhengzhou University, Zhengzhou China. http://www.ntu.edu.sg/home/epnsugan/

[2] Awad, N. H., Ali, M. Z. Suganthan, P. N., Reynolds, R. G., and Shatnawi, A. M. 2017. A Novel Differential Crossover Strategy based on Covariance Matrix Learning with Euclidean Neighborhood for Solving Real-World Problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, New York, USA, 380–386.

[3] Brest, J., Maučec, M. S., and Bošković, B. 2017. Single Objective Real-Parameter Optimization: Algorithm jSO. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, New York, USA, 1311–1318.

[4] Bujok, P. and Poláková, R. 2018. Migration Model of jSO Algorithm. In *2018 25th International Conference on Systems Signals and Image Processing (IWSSIP)*. IEEE Slovenia Sect; Univ Maribor, IEEE, New York, USA.

[5] Das, S., Mullick, S. S., and Suganthan, P. N. 2016. Recent advances in differential evolution-An updated survey. *Swarm and Evolutionary Computation* 27, pp. 1–30.

[6] Das, S. and Suganthan, P. N. 2011. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* 15, pp. 27–54.

[7] Guo S. M., Tsai, J. S.-H., Yang, C. C., and Hsu, P.-H. 2015. A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set. In *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, New York, USA, 1003–1010.

[8] Piotrowski A. P. and Napiorkowski J. J. 2018. Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure? *Swarm and Evolutionary Computation* 43, pp. 88–108. DOI: https://doi.org/10.1016/j.swevo.2018.03.007

[9] Poláková, R., Tvrdík, J., and Bujok P. 2019. Differential evolution with adaptive mechanism of population size according to current population diversity. *Swarm and Evolutionary Computation*, In Press. DOI: https://doi.org/10.1016/j.swevo.2019.03.014

[10] Storn, R. and Price, K. V. 1995. *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Int. Comp. Sci, Inst., Berkeley, CA. http://www.icsi.berkeley.edu/ storn/litera.html

[11] Storn, R. and Price, K. V. 1997. Differential evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, pp. 341–359.

[12] Wang Y., Li, H.-X., Huang T., and Li, L. 2014. Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Applied Soft Computing* 18, pp. 232–247.