Eigenvector Crossover in jDE100 Algorithm

Petr Bujok
Department of Informatics and Computers
Faculty of Science, University of Ostrava
30. dubna 22, 70200 Ostrava, Czech Republic
petr.bujok@osu.cz

Patrik Kolenovsky
Department of Informatics and Computers
Faculty of Science, University of Ostrava
30. dubna 22, 70200 Ostrava, Czech Republic
p19091@student.osu.cz

Vladimir Janisch
Department of Informatics and Computers
Faculty of Science, University of Ostrava
30. dubna 22, 70200 Ostrava, Czech Republic
p19031@student.osu.cz

***Abstract*—In this paper, an advanced variant of the efficient jDE100 variant is proposed. The introduced jDE100e employs an efficient rotationally-invariant Eigenvector crossover. Both algorithms are applied on 10 test problems for a single-objective optimisation CEC 2020 with four dimension levels. The results show that the proposed jDE100e is able to solve some of the problems successfully. A comparison of the proposed jDE100e with the original jDE100 illustrates the superiority of the newly designed algorithm as jDE100e performs significantly better in 16 out of 40 problems, and jDE100 is not able to outperform jDE100e significantly.**

***Index Terms*—Differential evolution, jDE100, Eigenvector crossover, experiments, test problems**

I. Introduction

Global optimisation is an evolving and widely applied research area. A single-objective real-parameter problem of global optimisation is typically limited by bound constraints and it is defined as follows. For the objective function, $f(\boldsymbol{x}),\ \boldsymbol{x} = (x_1,\ x_2, \ldots, x_D) \in \mathbb{R}^D$ and the bounded search area $\Omega = \prod_{j=1}^{D}[a_j,\ b_j],\ a_j < b_j,\ j = 1,\ 2, \ldots, D$, the global minimum (maximum) is such a point $\boldsymbol{x}^*$, which satisfies condition $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \Omega$ is the solution of the problem.

There are many approaches to solve global optimisation problems, for instance, methods inspired by evolution of species are very popular. These methods are called Evolutionary algorithms (EA) and achieve good efficiency and applicability. One of the main reasons of good results provided by the EA algorithms is development of a population of possible solutions using stochastic and heuristic operations. One of the most often developed and applied EA is called Differential evolution (DE). The DE algorithm is very simple because it uses only a small number of operations and control parameters, but it provides very good accuracy and efficiency when solving optimisation tasks from various areas [1]–[3].

In this paper, an enhanced variant of the adaptive DE algorithm is proposed. The best performing optimisation technique assessed at the last Congress on Evolutionary Computation (CEC 2019 [4]) was jDE100 [5]. Although the jDE100 algorithm uses only one mutation strategy and one crossover variant, it achieved the best results in all ten problems. The results and use of a simple crossover variant were the main motivation to design an enhanced version of jDE100 using covariance-based crossover type [6].

The rest of the paper is organised as follows. The basic idea of the Differential evolution algorithm is in Section II. An efficient jDE100 variant and Eigenvector crossover are briefly described in Section III and IV. The main idea of the proposed variant of the jDE100 algorithms is in Section V. Experimental settings and results are discussed in Section VI and VII. Finally, Section VII contains some concluding points on the proposed method.

II. Differential Evolution

In 1996, Storn and Price proposed a simple and efficient optimisation algorithm called Differential Evolution (DE) [7]. The main advantages of the DE algorithm are its simplicity and good efficiency. The main steps of the DE algorithm are depicted in a pseudo-code of Algorithm 1.

Algorithm 1 Differential evolution algorithm

```
initialise population $P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
while stopping condition not reached do
  for $i = 1, 2, \ldots, N$ do
    create a new trial vector $\boldsymbol{y}_i$
    if $f(\boldsymbol{y}_i) \leq f(\boldsymbol{x}_i)$ then
      insert $\boldsymbol{y}_i$ into $Q$
    else
      insert $\boldsymbol{x}_i$ into $Q$
    end if
  end for
  $P \leftarrow Q$
end while
```

At first, population of N individuals (potential solutions) is randomly generated in the search area and evaluated by the objective function f. Then, the development of the population is repeated until the stopping condition is satisfied. The individuals of the population are developed by evolutionary operators - mutation, crossover, and selection. A new point (offspring) $\boldsymbol{y}_i$ is generated from the parent point $\boldsymbol{x}_i$ as follows. A mutated point $\boldsymbol{v}_i$ is computed from the parent individual using mutation. The most popular mutation variant used in DE is called rand/1 (1), where $r1, r2, r3$ are randomly selected, but mutually distinct indices from $[1,\ N]$, different from i, and $F \in (0,\ 2]$ is a scale factor.

$$\boldsymbol{v} = \boldsymbol{x}_{r1} + F \cdot (\boldsymbol{x}_{r2} - \boldsymbol{x}_{r3}) \tag{1}$$

After mutation, elements of the parent vector $\boldsymbol{x}_i$ and mutated vector $\boldsymbol{v}_i$ are combined to a new trial point - $\boldsymbol{y}_i$. This operation is called crossover and the most popular is binomial crossover (2), which is controlled by crossover ratio $CR \in (0,\ 1)$.

$$y_{i,j} = \begin{cases} v_{i,j}, & if\ rand_j(0,1) \le CR\ or\ j = rand_j(1, D) \\ x_{i,j}, & otherwise. \end{cases} \tag{2}$$

A newly generated individual $\boldsymbol{y}_i$ is evaluated by an objective function and it is inserted into an auxiliary population Q if it is better than the parent individual $\boldsymbol{x}_i$, i.e. $f(\boldsymbol{y}_i) \le f(\boldsymbol{x}_i)$. Otherwise, the old solution is copied into the auxiliary population Q. At the end of generation, the auxiliary population Q replaces the old population P. The last operation is known as selection, and such a simple variant of DE is called standard or canonical DE.

The efficiency of the standard DE is worse when solving complex optimisation problems, or large scale problems with high dimension D. The problem is in fixed values of control parameters - N, CR, F. This results in adaptation of the control parameters values in DE to cope with various optimisation problems. In more than two decades of DE, many adaptive variants of this algorithm have been proposed and successfully applied to real problems [1]–[3].

III. Adaptive DE Variant - jDE100

One of very efficient and recently proposed adaptive DE variants is called jDE100 [5]. This method was introduced by Brest et al. in 2019, when jDE100 was the best performing optimisation algorithm in the CEC 2019 competition for single-objective optimisation problems.

In contrast to standard DE, the jDE100 algorithm employs several advanced approaches. The main steps of the jDE100 algorithm are in the pseudo-code of Algorithm 2. At first, independent populations - a big population P_b of size N_b and small population P_s of size N_s - are randomly initialised and evaluated. In addition, the control parameters of mutation and crossover are initialised, $F = 0.5$ and $CR = 0.9$ for each point in both populations. After initialisation, development of both populations is performed until the stopping condition. Before evaluation of populations, the conditions for reinitialisation of populations are checked. The big population P_b is reinitialised (all individuals are randomly generated) if *my_eps* percent of the best individuals from P_b is evaluated by similar objective function value (measured by parameter $\varepsilon < 1 \times 10^{-16}$). The big population is also reinitialised if the best point $\boldsymbol{x}_{\text{best}}$ is not improved in the *ageLmt* generations.

Algorithm 2 jDE100 algorithm

```
initialise big population $P_b = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_b}\}$
initialise small population $P_s = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_s}\}$
initialise $F_i = 0.5$, $CR_i = 0.9,\ i = 1, 2, \ldots (N_b + N_s)$
while stopping condition not reached do
  if reinitialisation condition is reached then
    reinitialise populations
  end if
  for $i = 1, 2, \ldots, N_b$ do
    create a new trial vector $\boldsymbol{y}_i$
    if $f(\boldsymbol{y}_i) \le f(\boldsymbol{x}_i)$ then
      insert $\boldsymbol{y}_i$ into $Q_b$
    else
      insert $\boldsymbol{x}_i$ into $Q_b$
    end if
  end for
  $P_b \leftarrow Q_b$
  if $\boldsymbol{x}_{\text{best}} \in P_b$ then
    $\boldsymbol{x}_{\text{best}} \rightarrow P_s$
  end if
  for $k = 1, 2, \ldots, m$ do
    for $j = 1, 2, \ldots, N_s$ do
      create a new trial vector $\boldsymbol{y}_j$
      if $f(\boldsymbol{y}_j) \le f(\boldsymbol{x}_j)$ then
        insert $\boldsymbol{y}_j$ into $Q_s$
      else
        insert $\boldsymbol{x}_j$ into $Q_s$
      end if
    end for
    $P_s \leftarrow Q_s$
  end for
end while
```

The small population P_s is reinitialised (all individuals are randomly generated) if *my_eps* percent of individuals from P_s is evaluated by a similar objective function value as the best point of P_s (measured by parameter $\varepsilon < 1 \times 10^{-16}$). Then, all points are randomly generated, except the best solution $\boldsymbol{x}_{\text{best}}$. Note, if the population is reinitialised, values of F_i and CR_i are also set to the initial values, i.e. $F_i = 0.5$ and $CR_i = 0.9$.

After that, one generation of the big population P_b is performed. A well-known rand/1 mutation is preferred, where index $r1$ is selected from P_b and indices $r2, r3$ are selected from a union $P_b \bigcup \boldsymbol{x}_s$, where $\boldsymbol{x}_s$ is the point from the small population. Further, binomial crossover and selection are applied to generate a new trial point. Similarly to canonical DE, auxiliary populations Q_B and Q_S are used to develop a new generation of individuals. At the end of generation in P_b, the position of the best point is checked - if it is located in P_b, then a copy of $\boldsymbol{x}_{\text{best}}$ is sent to the small population. Then m generations of the small population P_s are performed, $m = \text{round}(N_b/N_s)$. The process of evaluation of P_s is similar to the evaluation of P_b, the indices $r1, r2, r3$ are selected from P_s.

Note, jDE100 is derived from the original jDE [8], therefore adaptation of F, *CR* is also used. The values of these parameters are the same if the individual produced a good new trial solution. In other cases, F and *CR* are randomly changed in given intervals $(F_l,\ F_u)$ and $(CR_l,\ CR_u)$. A detail of the jDE100 algorithm is available in the original papers [5], [8].

IV. Eigenvector Coordinate System

Although jDE100 is a very efficient optimisation method, there is only a standard binomial crossover employed to generate trial individuals. It was shown that the efficiency of DE with binomial crossover with $CR < 1$ is rather poor when solving rotated objective functions [9]. Therefore, a variant of DE with rotationally-invariant crossover should be used.

In 2014, the covariance-based Eigenvector coordinate system for a crossover operation in DE (CoBiDE) was proposed [6]. The main aim of this approach was better efficiency in functions with strongly correlated coordinates. Simply, a covariance matrix C from a portion of population (*ps*) is computed. This matrix is used to compute Eigenvectors B and Eigenvalues E using Eigen decomposition:

$$C = BE^2B^T. \tag{3}$$

A mutated point $\boldsymbol{v}_i$ and the parent point $\boldsymbol{x}_i$ are combined in the Eigen coordinate system, with probability *pb*:

$$\boldsymbol{x}_i^{'} = B^T\boldsymbol{x}_i \qquad \text{and} \qquad \boldsymbol{v}_i^{'} = B^T\boldsymbol{v}_i. \tag{4}$$

After that, a newly developed trial vector $\boldsymbol{y}_i^{'}$ is transformed from the Eigen coordinate system back to a standard coordinate system using the same Eigenvectors B.

$$\boldsymbol{y}_i = B\boldsymbol{y}_i^{'} \tag{5}$$

This transformation of individuals in crossover is performed only with a probability *pb*. In other cases of generations, the standard binomial crossover operation in a standard coordinate system is preferred.

For simplicity, this type of crossover will be named Eigenvector crossover. The Eigenvector crossover was successfully used in a study where it achieved good results [10]. Moreover, it was shown that the Eigenvector crossover enables to increase the efficiency of very successful adaptive jSO [11]. These studies inspired to use the Eigenvector crossover in the very efficient jDE100 algorithm.

V. jDE100e - jDE100 with Eigenvector Crossover

The newly proposed variant of the original jDE100 algorithm is enhanced by the Eigenvector crossover from CoBiDE. Therefore, the new method is simply denoted jDE100e. The steps of jDE100e are illustrated in a pseudo-code of Algorithm 3. The main differences between the original jDE100 and new jDE100e are highlighted by arrows.

The main aim of the jDE100e algorithm is to employ the rotationally-invariant crossover method in the very efficient jDE100 method. At the beginning of generation, it is checked

Algorithm 3 jDE100e algorithm

```
initialise big population $P_b = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_b}\}$
initialise small population $P_s = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_s}\}$
initialise $F_i = 0.5,\ CR_i = 0.9,\ i = 1, 2, \ldots (N_b + N_s)$
while stopping condition not reached do
  if reinitialisation condition is reached then
    reinitialise populations
  end if
  if $rand < pb$ then
    for $i = 1, 2, \ldots, N_b$ do
      create $\boldsymbol{y}_i$ using Eigenvector crossover        $\Leftarrow$
      if $f(\boldsymbol{y}_i) \le f(\boldsymbol{x}_i)$ then
        insert $\boldsymbol{y}_i$ into $Q_b$
      else
        insert $\boldsymbol{x}_i$ into $Q_b$
      end if
    end for
  else
    for $i = 1, 2, \ldots, N_b$ do
      create $\boldsymbol{y}_i$ using binomial crossover        $\Leftarrow$
      if $f(\boldsymbol{y}_i) \le f(\boldsymbol{x}_i)$ then
        insert $\boldsymbol{y}_i$ into $Q_b$
      else
        insert $\boldsymbol{x}_i$ into $Q_b$
      end if
    end for
  end if
  $P_b \leftarrow Q_b$
  if $\boldsymbol{x}_{\text{best}} \in P_b$ then
    $\boldsymbol{x}_{\text{best}} \rightarrow P_s$
  end if
  for $k = 1, 2, \ldots, m$ do
    if $rand < pb$ then
      for $j = 1, 2, \ldots, N_s$ do
        create $\boldsymbol{y}_j$ using Eigenvector crossover        $\Leftarrow$
        if $f(\boldsymbol{y}_j) \le f(\boldsymbol{x}_j)$ then
          insert $\boldsymbol{y}_j$ into $Q_s$
        else
          insert $\boldsymbol{x}_j$ into $Q_s$
        end if
      end for
    else
      for $j = 1, 2, \ldots, N_s$ do
        create $\boldsymbol{y}_j$ using binomial crossover        $\Leftarrow$
        if $f(\boldsymbol{y}_j) \le f(\boldsymbol{x}_j)$ then
          insert $\boldsymbol{y}_j$ into $Q_s$
        else
          insert $\boldsymbol{x}_j$ into $Q_s$
        end if
      end for
    end if
    $P_s \leftarrow Q_s$
  end for
  update matrices $C$ and $B$        $\Leftarrow$
end while
```

whether the Eigenvector crossover will be used ($pb < rand$). Then, all individuals of a given population are generated using the new crossover variant. Otherwise, the standard binomial crossover is used for the whole population. This approach is used for P_b and also for P_s, independently. To reduce the complexity of the proposed method, the matrices C and B are updated only if the Eigenvector crossover is selected to be used (3).

VI. Setting of the Experiment

The newly proposed jDE100e and the original jDE100 algorithm were applied to a suite of 10 problems from the CEC 2020 [12]. This test suite contains problems with various complexity, and all ten problems are used with four dimension levels $D \in [5,\ 10,\ 15,\ 20]$. Therefore 40 problems are used as a benchmark in this experimental study. For each algorithm and test problem, 30 independent runs were performed. The search process of the algorithms is limited to a given number of function evaluations, *maxFES*. For this test suite, the maximum number of function evaluations varies for different D (Table I).

TABLE I
Maximal number of function evaluations per run.

D	maxFES
5	50000
10	1000000
15	3000000
20	10000000

Moreover, the best-achieved results of the algorithms are stored in 15 preliminary stages, i.e. when $D^{\frac{k}{5}-3}$ function evaluations are reached, $k = 0,\ 1,\ 2, \ldots 15$. The difference between the real solution of the problem and the solution provided by applied algorithm is denoted as an error, where $error = f(\boldsymbol{x}_{\text{best}}) - f(\boldsymbol{x}^*) = 0$ represents a successful run. The error value under $1\times^{-8}$ is taken as zero. Further, the authors of the CEC 2020 test problems recommended solving all problems except problems $\#6,\ \#7$ and $D = 5$. Therefore, the table of results for $D = 5$ contains only results from 9 test problems.

The parameters of the jDE100 algorithm were set to the values recommended by the authors of this algorithm, $N_b = 1000$, $N_s = 25$, $\varepsilon = 1 \times 10^{-16}$, $\tau_1 = \tau_2 = 0.1$, $myEps = 25$, $F_{\text{lb}} = 0.15$, $F_{\text{ls}} = 0.15$, $F_u = 1.1$, $CR_l = 0$, $CR_u = 1.1$, and $ageLmt = 0.75 \cdot maxFES$.

The proposed jDE100e differs only in applied Eigenvector crossover, which is controlled by two parameters - a portion of the population ps and the frequency of using the Eigen coordinate system pb. The authors of CoBiDE recommended using $ps = 0.5$ and $pb = 0.4$. After the application of several different settings of these parameters, the original settings were found the best performing in this experimental study. The remaining parameters of jDE100e were set to the recommended values by the authors of the original jDE100.

Both DE variants are implemented and experimentally tested in Matlab 2017b. All computations were carried out on a standard PC with Windows 10, Intel(R) Core(TM)i7-4790 CPU 3.6 GHz, 16 GB RAM.

VII. Results

The basic characteristics of the results of the proposed algorithm from the 40 various problems are illustrated in Table II, III, IV, and V. It is obvious that the proposed jDE100e is able to solve seven out of eight test problems for $D = 5$. In one problem, all runs were done with zero error values (problem #1). For $D = 10$, jDE100e solves successfully three out of ten problems, in two problems all runs were done with zero error values. Naturally, the efficiency of algorithms decreases with increasing dimensionality. Therefore, for $D = 15$, one problem is solved by jDE100e with zero error value in each of the performed runs. Finally, for the highest $D = 20$, zero error values are also observed for runs of problem #1.

TABLE II
Results of proposed jDE100e and $D = 5$

Func.	Best	Worst	Median	Mean	Std
1	0	0	0	0	0
2	0.124899	13.409	0.249798	1.342155	2.932142
3	0	5.39917	5.14823	4.518145	1.524361
4	0	0.152508	0.107173	0.086678	0.046698
5	0	0.994959	0	0.053969	0.211078
8	0	3.58492	0	0.119731	0.654471
9	0	100	100	93.33333	25.37081
10	0	347.367	300	289.5292	80.64704

TABLE III
Results of proposed jDE100e and $D = 10$

Func.	Best	Worst	Median	Mean	Std
1	0	0	0	0	0
2	0.217986	36.9483	7.077005	10.13692	10.16872
3	6.02955	15.287	11.8056	11.77447	1.830476
4	0.063911	0.233269	0.164405	0.16535	0.041797
5	0	1.98992	0.809694	0.815107	0.575156
6	0.012744	0.853188	0.348849	0.346238	0.222441
7	3.71E-07	0.027231	0.003157	0.0065	0.007873
8	0	100	0	33.33333	47.94633
9	100	329.544	100	107.6515	41.90881
10	397.743	398.009	397.743	397.7607	0.067486

TABLE IV
Results of proposed jDE100e and $D = 15$

Func.	Best	Worst	Median	Mean	Std
1	0	0	0	0	0
2	0.041637	10.1633	2.40161	2.595014	2.51594
3	15.567	17.0215	15.8493	16.01047	0.40246
4	0.128818	0.433341	0.257148	0.257412	0.069246
5	1.15108	9.26686	3.29711	3.876532	2.289827
6	0.185709	1.17642	0.473277	0.478963	0.212076
7	0.045567	0.528341	0.26391	0.265466	0.137924
8	100	100	100	100	0
9	100	390.308	389.678	312.1448	130.1184
10	400	400	400	400	0

Further, the time complexity of the proposed jDE100e algorithm is estimated by a given process. At first, standard

TABLE V
RESULTS OF PROPOSED jDE100E AND $D = 20$

Func.	Best	Worst	Median	Mean	Std
1	0	0	0	0	0
2	0.062457	5.21633	1.77	1.483665	1.496059
3	20.3872	21.8351	20.9675	20.97785	0.408856
4	0.116739	0.433288	0.365071	0.347499	0.080412
5	1.09904	4.29209	2.30217	2.370443	0.849243
6	0.069142	0.182842	0.111731	0.115486	0.03313
7	0.036796	0.423361	0.197886	0.214159	0.114189
8	100	100	100	100	0
9	399.781	407.07	404.9575	404.565	1.740636
10	399.041	413.657	413.657	412.9312	2.775854

computations are performed to estimate the current performance of the PC (T_0). After that, the complexity of the evaluation of test problem #7 is measured repeatedly (T_1). Finally, the complexity of the proposed method is estimated by a run of jDE100e on problem #7. This process is repeated five times to achieve average value time-complexity T_2. Then, the true time complexity is computed by $(T_2 - T_1)/T_0$. The authors of CEC 2020 recommended estimating the complexity only on $D = 5,\ 10,\ 15$. The achieved results of the complexity for proposed jDE100e method are illustrated in Table VI. It is obvious that the time complexity is increasing with increasing dimensionality, the high increase of jDE100e is mainly caused by the employed Eigenvector crossover.

TABLE VI
COMPUTATIONAL COMPLEXITY OF PROPOSED jDE100E

D	T_0	T_1	T_2	$(T_2 - T_1)/T_0$
5	0.1172	0.23	0.25	0.17
10	0.11	0.25	5.5	47.72
15	0.1329	0.34	5.82	41.23

The newly proposed jDE100e variant is derived from the original jDE100 algorithm. Therefore, a comparison of the algorithms is performed to show the difference between the optimisers. The non-parametric Wilcoxon rank-sum test was applied to both algorithms, for each problem and D independently. The difference between the methods is significant if the achieved significance level is less than 0.05. The results from this comparison are in Table VII. The median values for each algorithm, function and dimension is provided (columns 'jDE100' and 'jDE100e'). For better readability, a smaller median value for each problem is underlined. When the difference is statistically significant, the median value is printed in bold.

It is obvious that the original jDE100 algorithm performs better in two problems for $D = 10$, two problems for $D = 15$, and one problem for $D = 20$. However, these differences are not significant. On the other hand, the proposed jDE100e achieves a smaller (better) median value in 22 problems out of 40. The difference is significant in 16 problems, i.e. the achieved p-value from the Wilcoxon test is less than 0.05. Regarding dimensionality, jDE100e performs better than jDE100 in five problems for $D = 5$, six problems and $D = 10$,

TABLE VII
RESULTS OF THE WILCOXON RANK-SUM TEST.

D	Func.	jDE100	p-value	jDE100e
5	1	0	$\approx$	0
5	2	4.59914	0.000788	**0.249798**
5	3	5.596435	1.43E-06	**5.14823**
5	4	0.120696	0.111987	0.107173
5	5	0	0.394133	0
5	6	0	0.041774	**0**
5	8	0	0.356048	0
5	9	100	0.784066	100
5	10	344.995	0.000683	**300**
10	1	0	$\approx$	0
10	2	9.156015	0.307649	7.077005
10	3	11.57115	0.428957	11.7828
10	4	0.159046	0.190718	0.163426
10	5	5.39108	8.67E-07	**0.809694**
10	6	0.444535	0.067869	0.348849
10	7	0.327871	4.18E-09	**0.003157**
10	8	35.91645	0.021687	**0**
10	9	100	$\approx$	100
10	10	398.0325	3.66E-07	**397.743**
15	1	0	$\approx$	0
15	2	2.804225	0.028	**2.40161**
15	3	15.72265	0.220697	15.8493
15	4	0.26192	0.982307	0.257148
15	5	17.84065	5.23E-10	**3.29711**
15	6	0.33655	0.251881	0.473277
15	7	1.525685	3.02E-11	**0.26391**
15	8	100	$\approx$	100
15	9	389.96	4.00E-05	**389.678**
15	10	400	$\approx$	400
20	1	0	$\approx$	0
20	2	1.848075	0.14416	1.77
20	3	21.06915	0.678161	20.9675
20	4	0.347209	0.118817	0.365071
20	5	15.7861	1.49E-10	**2.30217**
20	6	0.238378	5.00E-09	**0.111731**
20	7	0.642076	2.02E-08	**0.197886**
20	8	100	$\approx$	100
20	9	405.93	0.028129	**404.9575**
20	10	413.657	0.610827	413.657
better		5		22
better sign.		0		16

five problems and $D = 15$, and six problems and $D = 20$. Note, in the case of problem #6, $D = 5$, the mean value is smaller for the jDE100e variant.

In jDE100e, successes of the binomial and Eigenvector crossover are studied. When a newly generated individual is better than the parent individual, the success of a currently applied crossover variant is increased by one. The average counts of successes of both crossover variants on each problem and dimension level are in Table VIII, where the success of binomial crossover is denoted by 'bin', and the success of the newly used Eigenvectors crossover is abbreviated 'Eig'. It is clear that the efficiency of the newly used Eigenvector crossover is rare for lower dimension levels. When dimensionality increases, the success of the rotationally-invariant crossover is rapidly higher compared with the success of binomial crossover.

VIII. CONCLUSION

In this paper, an advanced variant of very successful jDE100 algorithms is introduced. The proposed jDE100e employs

TABLE VIII
SUCCESS OF THE BINOMIAL AND EIGENVECTOR CROSSOVER IN JDE100E.

D	F	bin	Eig	D	F	bin	Eig
5	1	**3645**	3036	15	1	79523	**105122**
5	2	**3797**	2243	15	2	**96304**	34590
5	3	**6016**	3739	15	3	**103507**	58801
5	4	**6660**	4216	15	4	**44215**	14553
5	5	**4741**	3169	15	5	51303	**102176**
5	6	**3246**	2271	15	6	**139502**	59653
5	7	**3231**	2356	15	7	**167943**	127368
5	8	**5262**	3264	15	8	**172954**	116672
5	9	**7797**	4809	15	9	**114439**	67273
5	10	**9450**	7902	15	10	**185363**	127072
10	1	**10892**	10384	20	1	112771	**128042**
10	2	**13623**	5521	20	2	221919	**296346**
10	3	**19967**	10447	20	3	193698	**212817**
10	4	**23411**	9495	20	4	**73935**	23866
10	5	24072	**43353**	20	5	216100	**326995**
10	6	**43166**	32559	20	6	**604485**	361417
10	7	**49616**	39377	20	7	**332686**	290600
10	8	**31804**	24797	20	8	**208404**	144826
10	9	**134127**	91451	20	9	**73645**	39041
10	10	**71424**	62633	20	10	**156158**	93183

an efficient rotationally-invariant Eigenvector crossover. Both algorithms are applied to 40 test problems of the single-objective scenario CEC 2020 competition. The results of the proposed jDE100e are promising, as this method is able to solve eight problems for $D = 5$, three problems for $D = 10$, and one problem for $D = 15$ and $D = 20$.

The performance of jDE100e was compared with the results of the original jDE100. The results of the algorithms were assessed statistically using the non-parametric Wilcoxon rank-sum test. The original jDE100 performs better in five out of 40 problems, but the differences are not significant. The proposed jDE100e achieves better results in 22 out of 40 problems, in 16 cases the differences are significant. The proposed advanced variant of the successful jDE100 algorithms provides promising results compared with the original method. The efficiency of the newly used Eigenvector crossover increases with increasing dimensionality of the problems.

REFERENCES

[1] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, pp. 61–106, 2010.

[2] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, pp. 27–54, 2011.

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

[4] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization," Technical Report, Nanyang Technological University, Singapore, Tech. Rep., 2018, http://www.ntu.edu.sg/home/epnsugan/.

[5] J. Brest, M. S. Maučec, and B. Bošković, "The 100-digit challenge: Algorithm jDE100," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 19–26.

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

[7] R. Storn and K. V. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[8] J. Brest, S. Greiner, B. Boškovič, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 646–657, 2006.

[9] S.-M. Guo and C.-C. Yang, "Enhancing differential evolution utilizing eigenvector-based crossover operator," *IEEE Transactions on Evolutionary Computation*, vol. 19, pp. 31–49, 2015.

[10] P. Bujok and R. Poláková, "Eigenvector crossover in the efficient jSO algorithm," *MENDEL*, vol. 25, no. 1, pp. 65–72, Jun. 2019.

[11] P. Bujok, "Competition of strategies in jso algorithm," in *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing*, A. Zamuda, S. Das, P. N. Suganthan, and B. K. Panigrahi, Eds. Cham: Springer, 2020, pp. 113–121.

[12] C. T. Yue, K. V. Price, P. N. Suganthan, J. J. Liang, M. Z. Ali, B. Y. Qu, N. H. Awad, , and P. P. Biswas, "Problem definitions and evaluation criteria for the cec 2020 special session and competition on single objective bound constrained numerical optimization," Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China And Technical Report, Nanyang Technological University, Singapore, Tech. Rep., 2019, http://www.ntu.edu.sg/home/epnsugan/.