Differential Evolution Algorithm for Single Objective Bound-Constrained Optimization: Algorithm j2020

Janez Brest, Mirjam Sepesy Maučec, Borko Bošković
Faculty of Electrical Engineering and Computer Science
University of Maribor
Koroška cesta 46, 2000 Maribor, Slovenia
Email: janez.brest@um.si, mirjam.sepesy@um.si, borko.boskovic@um.si

***Abstract*—In this paper, a new algorithm is presented to deal with real parameter single-objective optimization problems, which are often complex and computationally very expensive. The proposed algorithm (j2020) is based on the self-adaptive differential evolution algorithms jDE and jDE100. Our algorithm uses two populations like jDE100, while jDE uses only one population. It uses a crowding mechanism, which is not being used in previous algorithms, and a mechanism to choose vectors in the mutation operation from both subpopulations. We provide the obtained results for each benchmark function for four dimension scenarios as required by the organizers of the special session for Single Objective Bound-Constrained Optimization. We also compare the obtained results with the original DE and jSO algorithms on the largest dimension scenario.**

***Index Terms*—differential evolution, optimization, global optimum, maximum number of function evaluations**

I. Introduction

Single-objective real parameter optimization is interesting since we can find it in many practical problems in various research domains. Also at CEC, the competition and/or special session has been organized each year after 2013. Solving complex optimization problems, either they are single-objective or not, is a challenging task because a search space is huge even for a small number of objectives and it grows tremendously when the number of objectives is increasing.

It is known that one of the biggest drawbacks of evolutionary algorithms, as well as other population-based algorithms, like particle swarm optimization (PSO) algorithms, is the loss of diversity in the population. As a consequence, an algorithm might show a premature convergence into local optima [1]. On the other hand, population-based algorithms have some advantages when dealing with ways to solve single-objective real parameter optimization, among others, they do not need any gradient calculation.

A goal in the global optimization problem is to find a solution, i.e., vector $\vec{x}$, which minimizes objective function $f(\vec{x})$. Vector $\vec{x} = \{x_1, x_2, ..., x_D\}$ consists of D variables, and each variable $x_j, j = 1, 2, ..., D$ is defined by its lower $x_{j,low}$ and upper $x_{j,upp}$ bound. Therefore, we can define a global optimization problem also as bound-constrained optimization where D denotes the dimensionality of the problem.

In a single objective optimization, function f might have many optima and an algorithm is required to find the global one. If an algorithm traps its population into a local optimum, the final result of optimization may be poor [2]. The global optimum may not be known, which is a case when we are solving an unknown problem for the first time.

The Differential Evolution (DE) [3] was proposed by R. Storn and K. Price 35 years ago. This stochastic population-based algorithm has shown competitive performances when solving real-world optimization problems [4], [5] in various domains.

At CEC 2019 competition on the 100-Digit Challenge [6], some DE versions [7], [8], [1] were among top algorithms (see Analysis of Results [9]) among other optimization algorithms like SOMA [10], ABC [11].

Nowadays, a lot of improvements on DE were proposed recently, and they were collected in reviews and surveys [12], [13], [14], [15], [16].

In this paper, we present a new version of the DE algorithm, called j2020, for single objective real parameter optimization [17]. It is a derivation from the jDE [18] and jDE100 [1] algorithms. The main new features of the j2020 algorithm are:

(1) It uses two populations.
(2) Lower and upper limits of the control parameters are set differently compared to original jDE.
(3) It uses a simple one-way migration of currently best individual among the populations.
(4) It applies a mechanism that individuals from a small population might participate in the mutation operation of a big population.
(5) The proposed algorithm applies restart's mechanism separately in both populations to manage a population diversity.
(6) A crowing mechanism is used.

The ideas of these features/mechanisms are not novel in evolutionary algorithms and other optimization algorithms, some of them were already used in jDE versions, etc. Nevertheless, an algorithm, which joins more mechanisms under

The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0041, and P2-0069).

978-1-7281-6929-3/20/$31.00 ©2020 European Union

Require: **P**... the population
Require: NP ... the population size
Ensure: $\vec{x}_{best}$ ... the best individual in the population
Ensure: $f(\vec{x}_{best})$ ... the value of the best individual

Initialize $F_i = 0.5$; $CR_i = 0.9$; $(i \in \{1, NP\})$
Initialize population $\mathbf{P} = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{NP})$ randomly
while stopping criteria is not met **do**
  for $(i \leftarrow 0;\ i < NP;\ i{+}{+})$ **do**
    ▷ ** jDE mutation **
    $F \leftarrow \begin{cases} F_l + rand_1 * F_u, & \text{if } rand_2 < \tau_1, \\ F_{i,g}, & \text{otherwise,} \end{cases}$
    Randomly select r_1, r_2, and r_3; ▷ $r_1 \neq r_2 \neq r_3 \neq i$
    $\vec{v}_{i,g+1} \leftarrow \vec{x}_{r_1} + F(\vec{x}_{r_2} - \vec{x}_{r_3})$ ▷ DE/rand/1/
    ▷ ** jDE crossover **
    $CR \leftarrow \begin{cases} rand_3, & \text{if } rand_4 < \tau_2, \\ CR_{i,g}, & \text{otherwise,} \end{cases}$
    $j_{rand} \leftarrow rand\{1, D\}$
    for $(j \leftarrow 0;\ j < D;\ j{+}{+})$ **do**
      if $(rand(0,1) \leq CR$ **or** $j = j_{rand})$ **then**
        $u_{i,j,g+1} \leftarrow v_{i,j,g+1}$
      else
        $u_{i,j,g+1} \leftarrow x_{i,j,g}$
      end if
    end for
    ▷ ** selection **
    if $(f(\vec{u}_{i,g+1}) \leq f(\vec{x}_{i,g}))$ **then** ▷ minimization
      $\vec{x}_{i,g+1} \leftarrow \vec{u}_{i,g+1}$
      $F_{i,g+1} \leftarrow F$
      $CR_{i,g+1} \leftarrow CR$
    else
      $\vec{x}_{i,g+1} \leftarrow \vec{x}_{i,g}$
      $F_{i,g+1} \leftarrow F_{i,g}$
      $CR_{i,g+1} \leftarrow CR_{i,g}$
    end if
  end for
end while

Algorithm 1: jDE [18].

one umbrella and does this in a way that a good overall performance is achieved, could be called as a new one.

The contributions of this paper are: A new version of the algorithm (j2020) is presented, experiments for CEC 2020 competition on the single objective real parameter are conducted, the obtained results are presented in a form required by the competition organizers, and the obtained results are compared with the results of the DE and jSO algorithms.

The structure of the paper is as follows. Section II gives a background for this work, where an overview of the DE, jDE, and jDE100 algorithms is presented. Section III presents our new algorithm, called j2020, which is used for solving single-objective problems. Experimental results of the j2020 algorithm on CEC 2020 benchmark problems are presented in Section IV. Section V concludes the paper with some final remarks.

II. Background

This section gives some backgrounds on DE and its two variants jDE and jDE100.

A. Differential Evolution

Differential evolution (DE) [3] is a population-based algorithm that belongs to the group of evolutionary algorithms. It shows highly competitive performances in many applications dealing with optimization [19], [20], [21], [22], [23].

The original DE algorithm has three control parameters, scaling factor F, crossover parameter CR, and population size NP. These parameters are required to be set before the actual evolutionary process starts. In the original DE algorithm, they have the same fixed value during the whole evolutionary process. In the DE algorithm, a population $\mathbf{P}$ consists of NP individuals or vectors:

$$\mathbf{P}_g = (\vec{x}_{1,g}, \ldots, \vec{x}_{i,g}, \ldots, \vec{x}_{NP,g}),\ i = 1, 2, \ldots, NP,$$

where g is a generation index, $g = 1, 2, \ldots, G_{MAX}$. Each vector

$$\vec{x}_{i,g} = \{x_{i,1,g}, x_{i,2,g}, \ldots, x_{i,D,g}\}$$

consists of D variables and represents the solution of an optimization problem.

DE evolves a randomly initialized population throughout G_{MAX} generations guided the individuals in the searching process toward a global optimum. The best-found vector in the evolutionary process and its function value are returned as a final solution.

Initialization: A population is randomly initialized before the evolutionary process starts. Each vector gets randomly generated uniformly distributed values between lower and upper bound for all its components:

$$x_{i,j,0} = x_{j,low} + rand() * (x_{j,upp} - x_{j,low}).$$

During each generation, the population is evolved, and DE employs three operations for each individual, namely mutation, crossover, and selection. They are described in the next subsections.

Mutation: A mutant vector $\vec{v}_{i,g+1}$ is created using one of the mutation strategies. The 'DE/rand/1' mutation strategy has been introduced in the original DE algorithm [3] and it is a very often used mutation strategy in DE. This strategy randomly selects two vectors and their difference is multiplied by scale factor F and added to the third randomly selected vector:

$$\vec{v}_{i,g+1} = \vec{x}_{r_1,g} + F \cdot (\vec{x}_{r_2,g} - \vec{x}_{r_3,g}),$$

where r_1, r_2, and r_3 are randomly chosen indices within a set of $\{1, ..., NP\}$. They are pairwise different and also different from index i:

$$r_1 \neq r_2 \neq r_3 \neq i.$$

The other widely used DE mutation strategies are [24], [25]:

- "DE/best/1": $\vec{v}_{i,g+1} = \vec{x}_{best} + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$,
- "DE/current to best/1":
  $\vec{v}_{i,g+1} = \vec{x}_{i,g} + F(\vec{x}_{best} - \vec{x}_{i,g}) + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$,

- "DE/best/2":
  $\vec{v}_{i,g+1} = \vec{x}_{best} + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) + F(\vec{x}_{r_3,g} - \vec{x}_{r_4,g})$,
- "DE/rand/2":
  $\vec{v}_{i,g+1} = \vec{x}_{r_1,g} + F(\vec{x}_{r_2,g} - \vec{x}_{r_3.g}) + F(\vec{x}_{r_4,g} - \vec{x}_{r_5,g})$,
- "DE/current-to-pBest/1":
  $\vec{v}_{i,g+1} = \vec{x}_{i,g} + F(\vec{x}_{pBest} - \vec{x}_{i,g}) + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$,

where the indices r_1–r_5 represent the random and mutually different integers generated within the set $\{1, ..., NP\}$ and also different from index i. $\vec{x}_{best}$ is the best vector in a current generation, while $\vec{x}_{pBest}$ denotes one of the good individuals from the top $p\%$ individuals.

Each strategy has a different ability to maintain the population diversity which might increase/decrease algorithm's convergence rate during the evolutionary process.

Crossover: A mutant vector $\vec{v}_{i,g+1}$ generated by one of the mutation strategies is used in the next operation, called crossover. Binomial crossover is widely used in DE. Another type of crossover is exponential [3], [4]. The binomial crossover creates a trial vector $\vec{u}_{i,g+1}$ as follows:

$$u_{i,j,g+1} = \begin{cases} v_{i,j,g+1}, & \text{if } rand(0,1) \leq CR \text{ or } j = j_{rand}, \\ x_{i,j,g}, & \text{otherwise,} \end{cases}$$

for $i = 1, 2, ..., NP$ and $j = 1, 2, ..., D$. $CR \in [0, 1]$ is crossover parameter and presents the probability of creating components for a trial vector from a mutant vector. If a component was not selected from the mutant vector, then it is taken from the parent vector $\vec{x}_{i,g}$. Randomly chosen index $j_{rand} \in \{1, 2, ..., NP\}$ is responsible for the trial vector to contain at least one component from the mutant vector.

If some variables from the trial vector are out of bounds, a repair mechanism is applied. Our algorithm reflects these variables back into the search space.

Selection: After the crossover operation, the trial vector is evaluated – an objective function $f(\vec{u}_{i,g+1})$ is calculated. Then selection operation compares two vectors, population vector $\vec{x}_{i,g}$ and its corresponding trial vector $\vec{u}_{i,g+1}$, according to their objective function values. The better vector will become a member of the next generation. The selection operation for a minimization optimization problem is defined as follows:

$$\vec{x}_{i,g+1} = \begin{cases} \vec{u}_{i,g+1}, & \text{if } f(\vec{u}_{i,g+1}) \leq f(\vec{x}_{i,g}), \\ \vec{x}_{i,g}, & \text{otherwise.} \end{cases}$$

This selection operation is greedy and it is well-known for DE, but based on our knowledge it is rarely applied in other EAs.

Stopping condition: DE stops an evolutionary process after G_{MAX} generations. The stopping criteria can be expressed also by the maximum number of function evaluations, the maximum number of generations, time limit, etc.

Different settings of the control parameters can lead to a faster convergence of the DE algorithm. Therefore, a procedure of tuning control parameters takes place before the actual optimization process starts. A good tuning procedure can take a lot of time. Consequently, adaptive and self-adaptive mechanisms appear for adjusting control parameters during the evolutionary process. In early stages of the evolutionary process, values for control parameters may differ from values that are the most suitable at the mid-stages of the evolutionary process, and those at late stages.

Adaptive and self-adaptive approaches can be applied to

Require: $\mathbf{P_b}$ ... big population
Require: $\mathbf{P_s}$ ... small population
Require: *bNP* ... size of $\mathbf{P_b}$
Require: *sNP* ... size of $\mathbf{P_s}$: $bNP = m \times sNP, m \in 1, 2, ...$

Initialize population $\mathbf{P_b} = (\vec{x}_1, \ldots, \vec{x}_{bNP})$ randomly
Initialize population $\mathbf{P_s} = (\vec{x}_1, \ldots, \vec{x}_{sNP})$ randomly
Initialize $F_i = 0.5$; $CR_i = 0.9$; $(i \in \{1, bNP{+}sNP\})$
while stopping criteria is not met **do**
  Check for a reinitialization of $\mathbf{P_b}$
  Check for a reinitialization of $\mathbf{P_s}$
  for each $i \in \mathbf{P_b}$ **do**
    ▷ ** on big population **
    Perform **jDE mutation** (Steps 5–7 in Alg. 1)
    Perform **jDE crossover** (Steps 8–16 in Alg. 1)
    Perform **jDE selection** (Steps 17–25 in Alg. 1)
  end for
  if $\vec{x}_{best} \in \mathbf{P_b}$ **then**
    Copy $\vec{x}_{best}$ into $\mathbf{P_s}$
  end if
  for $k \in 1, 2, ..., m$ **do** ▷ repeat m-times
    for each $i \in \mathbf{P_s}$ **do**
      ▷ ** on small population **
      Perform **jDE mutation** (Steps 5–7 in Alg. 1)
      Perform **jDE crossover** (Steps 8–16 in Alg. 1)
      Perform **jDE selection** (Steps 17–25 in Alg. 1)
    end for
  end for
end while

Ensure: $\vec{x}_{best}$ ▷ the best individual in the population
Ensure: $f(\vec{x}_{best})$ ▷ the value of the best individual

Algorithm 2: jDE100 [1].

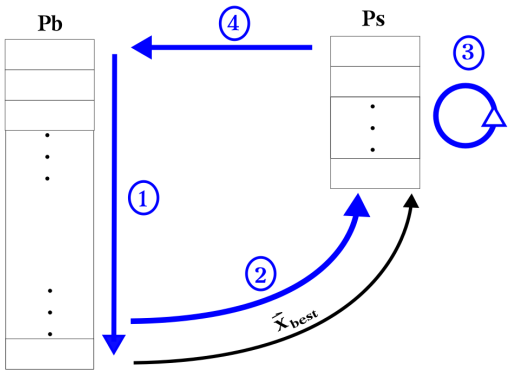

Fig. 1. Evolutionary process in the jDE100 [1] algorithm. Four steps are being repeated: ① One generation is performed on big population $\mathbf{P_b}$. ② If required, $\vec{x}_{best}$ is copied into small population $\mathbf{P_s}$. ③ More generations are performed on $\mathbf{P_s}$. ④ The evolutionary process control switches to $\mathbf{P_b}$.

Require: $\mathbf{P_b}$ ... big population
Require: $\mathbf{P_s}$ ... small population
Require: *bNP* ... size of $\mathbf{P_b}$
Require: *sNP* ... size of $\mathbf{P_s}$
Require: $bNP \geq sNP$ **and** $bNP = m \times sNP, m \in 1, 2, ...$

Initialize population $\mathbf{P_b} = (\vec{x}_1, \ldots, \vec{x}_{bNP})$ randomly
Initialize population $\mathbf{P_s} = (\vec{x}_1, \ldots, \vec{x}_{sNP})$ randomly
Initialize $F_i = 0.5$; $CR_i = 0.9$; $(i \in \{1, bNP{+}sNP\})$
while stopping criteria is not met **do**
  Check for a reinitialization of $\mathbf{P_b}$
  Check for a reinitialization of $\mathbf{P_s}$
  for each $i \in \mathbf{P_b}$ **do**
    ▷ ** on big population **
    perform jDE mutation (Steps 5–7 in Alg. 1) **using**
      $\vec{x}_{r_1} \in \mathbf{P_b}$
      $\vec{x}_{r_2}, \vec{x}_{r_3} \in \mathbf{P_b} \bigcup \mathbf{M_s}, \mathbf{M_s} \subset \mathbf{P_s}$
    end perform
    Perform **jDE crossover** (Steps 8–16 in Alg. 1)
    Perform crowding in $\mathbf{P_b}$
    Perform **jDE selection** (Steps 17–25 in Alg. 1)
  end for
  if $\vec{x}_{best} \in \mathbf{P_b}$ **then**
    Copy $\vec{x}_{best}$ into $\mathbf{P_s}$
  end if
  for $k \in 1, 2, ..., m$ **do** ▷ repeat m-times
    for each $i \in \mathbf{P_s}$ **do**
      ▷ ** on small population **
      Perform **jDE mutation** (Steps 5–7 in Alg. 1)
      Perform **jDE crossover** (Steps 8–16 in Alg. 1)
      Perform **jDE selection** (Steps 17–25 in Alg. 1)
    end for
  end for
end while

Ensure: $\vec{x}_{best}$ ▷ the best individual in the population
Ensure: $f(\vec{x}_{best})$ ▷ the value of the best individual

Algorithm 3: j2020.

a particular DE control parameter or more of them at the same time. Some proposed adaptive and self-adaptive DE approaches are jSO [2], SADE [26], jDE [18], JADE [27], LSHADE [28], [29], etc. These state-of-the-art algorithms utilize a self-adaptive mechanism for scale factor and crossover parameter, usually. Recently, there have been several attempts to adjust the third control parameter, i.e. population size (NP), during the evolutionary process [28], [30], but many researchers keep NP fixed during the optimization process.

B. jDE Algorithm

The jDE algorithm was introduced in 2006 [18]. It uses the self-adapting mechanism of two control parameters, i.e., scale factor and crossover rate. Each individual has its own control parameter values F_i and CR_i. New control parameters $F_{i,g+1}$ and $CR_{i,g+1}$ are calculated before the mutation operation is performed as follows [18]:

$$F_{i,g+1} = \begin{cases} F_l + rand_1 * F_u, & \text{if } rand_2 < \tau_1, \\ F_{i,g}, & \text{otherwise,} \end{cases}$$

$$CR_{i,g+1} = \begin{cases} rand_3, & \text{if } rand_4 < \tau_2, \\ CR_{i,g}, & \text{otherwise,} \end{cases}$$

where $rand_j$, for $j \in \{1, 2, 3, 4\}$ are random values uniformly distributed within the range $[0, 1]$. It can be seen that τ_1 and τ_2 values represent probabilities to adjust control parameters F and CR, respectively. For the sake of clarity, a pseudo-code of the jDE algorithm is depicted in Alg. 1.

In [18] and in many of our later works, parameters τ_1, τ_2, F_l, F_u are fixed to values $0.1, 0.1, 0.1, 0.9$, respectively. In such a way, the new F takes a value from $[0.1, 1.0]$, and the new CR from $[0, 1]$.

Commonly, CR value is set between 0 and 1. Also, in the case of self-adaptation in jDE, where its value changes through iterations, it is kept between these two limits. The LSGOjDE [25] algorithm keeps the same lower limit, but defines a different upper limit for each of three mutation strategies:

$$CR_{i,g+1} = \begin{cases} CR_l + rand_3 \cdot CR_u & \text{if } rand_4 < \tau_2, \\ CR_{i,g} & \text{otherwise.} \end{cases} \quad (1)$$

The lower limit CR_l is always set to 0. The value of the upper limit depends on the strategy. In the LSGOjDE, the upper limit is set to $0.25, 1.2$, and 1.2, respectively. In jDE100 [1], the F_l is a tunable parameter (it has been tuned for each benchmark function).

Note that $F_{i,g+1}$ and $CR_{i,g+1}$ are obtained before the mutation is performed. So they influence the mutation, crossover, and selection operations of the new vector $\vec{x}_{i,g+1}$.

C. Algorithm jDE100

In this section, we give an overview of the algorithm jDE100[1] that was presented previous year at the CEC conference. It is suitable for solving single-objective real parameter optimization. It uses two not equal-sized populations, i.e., a big population $\mathbf{P_b}$, and a small population $\mathbf{P_s}$. Figure 1 depicts the evolutionary process over these two populations. The sizes of two populations are $bNP = 1000$ and $sNP = 25$, respectively. The pseudo-code of jDE100 is given in Algorithm 2.

Population $\mathbf{P_b}$ is reinitialized once per generation if $myEqs = 25\%$ of bNP individuals have the similar value as the best individual in $\mathbf{P_b}$, or when the best individual in $\mathbf{P_b}$ is not improved for $ageLmt = 1e9$ evaluations that were applied only on $\mathbf{P_b}$. During reinitialization, all individuals are randomly generated like during the initialization process, except the $\vec{x}_{best}$ vector in $\mathbf{P_s}$ only.

Information exchange between the populations is passed by coping the best vector $\vec{x}_{best}$ from $\mathbf{P_b}$ to $\mathbf{P_s}$ if it has been found in $\mathbf{P_b}$, and during a mutation process. The details are presented in [1] and the source code is available at https://github.com/P-N-Suganthan.

Note that there was no limit of the maximum number of function evaluations in the CEC 2019 competition, while such a limit is set for each function at CEC 2020.

TABLE I
SUMMARY OF THE CEC 2020 BOUND-CONSTRAINED BENCHMARK FUNCTIONS.

	No.	Functions	$F_i^* = F_i(\vec{x}^*)$
Unimodal Functions	1	Shifted and Rotated Bent Cigar Function (CEC 2017 F1)	100
Basic Functions	2	Shifted and Rotated Schwefel's Function (CEC 2014 F11)	1100
	3	Shifted and Rotated Lunacek bi-Rastrigin Function (CEC 2017 F7)	700
	4	Expanded Rosenbrock's plus Griewangk's Function (CEC2017 $f19$)	1900
Hybrid Functions	5	Hybrid Function 1 (N = 3) (CEC 2014 F17)	1700
	6	Hybrid Function 2 (N = 4) (CEC 2017 F16)	1600
	7	Hybrid Function 3 (N = 5) (CEC 2014 F21)	2100
Composition Functions	8	Composition Function 1 (N = 3) (CEC 2017 F22)	2200
	9	Composition Function 2 (N = 4) (CEC 2017 F24)	2400
	10	Composition Function 3 (N = 5) (CEC 2017 F25)	2500
Search range: [-100,100]^D, $D = 5, 10, 15,$ and 20			

TABLE II
PARAMETERS IN THE J2020 ALGORITHM.

Parameter	Value	Description
bNP	$7D$	size of $\mathbf{P_b}$
sNP	D	size of $\mathbf{P_s}$
$F_{l,b}$	0.01	lower limit of scale factor for $\mathbf{P_b}$
$F_{l,s}$	0.17	lower limit of scale factor for $\mathbf{P_s}$
F_u	1.1	upper limit of scale factor for $\mathbf{P_b}$ and $\mathbf{P_s}$
$CR_{l,b}$	0.0	lower limit of crossover parameter for $\mathbf{P_b}$
$CR_{l,s}$	0.0	lower limit of crossover parameter for $\mathbf{P_s}$
$CR_{u,b}$	1.0	upper limit of crossover parameter for $\mathbf{P_b}$
$CR_{u,s}$	0.7	upper limit of crossover parameter for $\mathbf{P_s}$
F_{init}	0.5	initial value of scale factor
CR_{init}	0.9	initial value of crossover parameter
τ_1	0.1	probability to self-adapt scale factor
τ_2	0.1	probability to self-adapt crossover parameter
$ageLmt$	$maxFEs/10$	number of FEs with no improvement of the best individual then restart in $\mathbf{P_b}$ is required to occurs
eps	$1e-16$	small value used to check if two function values are similar
$myEqs$	25	reinitialization if $myEqs\%$ of individuals in the corresponding population have the similar function values

TABLE III
RESULTS FOR 5D.

Func.	Best	Worst	Median	Mean	Std
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0002	13.4090	0.5071	3.2283	3.7395
3	0.0000	6.0191	5.1483	3.4156	2.3271
4	0.0000	0.1912	0.1009	0.0768	0.0640
5	0.0000	0.9950	0.0000	0.1373	0.2860
8	0.0000	9.4160	0.0000	0.6278	2.3889
9	0.0000	100.0000	0.0511	20.4867	37.5459
10	0.0000	300.0000	100.1350	126.2370	90.3120

III. ALGORITHM J2020

A new algorithm (called j2020) for solving single-objective bound-constrained optimization is presented in this section. The j2020 pseudo-code is shown in Algorithm 3.

It uses the self-adaptation of F and CR control parameter applied at the individual's level and this mechanism is the same as in the jDE [18] algorithm.

The new algorithm uses two populations: a big population $\mathbf{P_b}$, and a small population $\mathbf{P_s}$, which is a similar approach as in the jDE100 [1] algorithm. Figure 1 illustrates the evolutionary process over these two populations in jDE100 and the same steps are applied in j2020. The sizes of the populations are bNP and sNP, respectively. Before the evolutionary process starts, the populations are initialized (Steps 1 and 2 in Alg. 3), and also the control parameters F_i and CR_i for both populations are initialized. The main loop represents the evolutionary process, which is iterated until stopping criteria is met. The stopping criteria is defined with a predefined number of function evaluations, $maxFEs$.

TABLE IV
RESULTS FOR 10D.

Func.	Best	Worst	Median	Mean	Std
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	3.6648	0.2498	0.6786	1.1601
3	0.0000	11.0110	10.3669	8.0587	3.8839
4	0.0000	0.2465	0.1282	0.1093	0.0904
5	0.0000	1.2031	0.2081	0.3022	0.3130
6	0.0291	0.8714	0.4845	0.4776	0.2491
7	0.0000	0.3204	0.0051	0.0673	0.1251
8	0.0000	11.5631	0.0000	1.5417	3.9979
9	0.0000	100.0000	100.0000	80.0000	40.6838
10	100.0069	397.7429	100.0503	140.1574	81.1932

TABLE V
RESULTS FOR 15D.

Func.	Best	Worst	Median	Mean	Std
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.1665	0.0416	0.0572	0.0432
3	0.0000	15.5670	0.0000	6.7789	7.8184
4	0.0000	0.3088	0.2055	0.1987	0.0747
5	0.0000	32.1552	5.4431	7.5816	7.6857
6	0.0016	8.6239	0.3371	0.8451	2.0921
7	0.0681	11.6390	0.6521	0.9828	2.0273
8	0.0000	100.0000	0.0000	9.4910	27.4228
9	100.0000	300.0000	100.0000	123.3855	56.8458
10	100.0212	400.0000	400.0000	390.0007	54.7684

Next two steps are responsible for checking if any of two populations needs to be reinitialized. Population $\mathbf{P_b}$ is reinitialized if $myEqs$ percent of the best individuals in $\mathbf{P_b}$ have a similar function value (a difference is less than a small value $eps = 1e-16$). The reinitialization is also applied if the best individual in $\mathbf{P_b}$ is not improved for $ageLmt$ evaluations. During the reinitialization process of the big population $\mathbf{P_b}$ all individuals of $\mathbf{P_b}$ are randomly reinitialized, i.e., each individual gets randomly generated uniformly distributed values between lower and upper bound for all its components.

Reinitialization in $\mathbf{P_s}$ has occurred if $myEqs\%$ of sNP individuals have a similar function value (a difference is less than a small value $eps = 1e-16$) as the best individual in $\mathbf{P_s}$. It reinitializes all individuals in $\mathbf{P_s}$ except the $\vec{x}_{best}$ vector in $\mathbf{P_s}$, which remains unchanged.

One generation is performed on the big population $\mathbf{P_b}$ (Lines 7–15 in Alg. 3). Mutant vector $\vec{v}_i$ is generated with well-known the jDE/rand/1 mutation strategy. Note, r_1 is the index of individual from $\mathbf{P_b}$, while r_2 and r_3 are indices of individuals from $\mathbf{P_b} \cup \mathbf{M_s}$, where $\mathbf{M_s} \subset \mathbf{P_s}$ presents a small amount of individuals from the small population $\mathbf{P_s}$. The motivation of using the proposed ranges for the indices r_2 and r_3 is to have a small influence of $\mathbf{P_s}$ on $\mathbf{P_b}$. The

TABLE VI
RESULTS FOR 20D.

Func.	Best	Worst	Median	Mean	Std
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0625	0.0312	0.0260	0.0247
3	0.0000	20.3872	20.3872	14.4196	9.2898
4	0.0298	0.3500	0.1732	0.1800	0.0784
5	0.3123	167.7516	81.4398	77.7693	57.4785
6	0.0684	0.4616	0.1769	0.1915	0.1013
7	0.0195	17.1125	0.3961	1.9843	4.0232
8	0.0000	100.0000	100.0000	92.7213	22.1020
9	100.0000	416.1669	405.9567	339.4512	127.5505
10	399.0080	399.1628	399.0493	399.0631	0.0402

TABLE VII
COMPUTATION COMPLEXITY. TIME IS PRESENTED IN SECONDS.

	T0	T1	*T2*	*(T2 - T1)/T0*
D =5	0	0.04657	0.1818	inf
D =10	0	0.07947	0.3327	inf
D =15	0	0.1302	0.5186	inf

number of individuals in $\mathbf{M_s}$ is expressed as follows:

$$|\mathbf{M_s}| = \begin{cases} 1, & \textbf{if } 1 \leq it \leq \frac{1}{3}MaxFEs, \\ 2, & \textbf{if } \frac{1}{3}MaxFEs < it \leq \frac{2}{3}MaxFEs, \\ 3, & \textbf{if } \frac{2}{3}MaxFEs < it \leq MaxFEs, \end{cases}$$

where $it = 1, \cdots, MaxFEs$ is the iteration counter.

It is expected that $\mathbf{P_s}$ may have a faster convergence speed (it also may get trapped into a local minimum) since sNP is smaller than bNP. Then the crossover and selection operations follow, which are identical as in jDE.

When one generation in $\mathbf{P_b}$ has finished, the algorithm checks if $\vec{x}_{best}$ was found in $\mathbf{P_b}$. If yes, then it is copied into the small population $\mathbf{P_s}$. In this way, a bigger influence of $\mathbf{P_b}$ on $\mathbf{P_s}$ is applied, and we want that the smaller population continues the evolutionary process also with one fresh (i.e. the best) individual.

Then mutation, crossover, and selection operations follow on smaller population $\mathbf{P_s}$ and they are similar to those on bigger population $\mathbf{P_b}$. The operations are repeated m-times, which ensures that both populations have an equal amount of function evaluations. In a special case, we can assume that $bNP = m \times sNP, m \in 1, 2, ...$ Note, when a mutation operation is applied to the smaller population $\mathbf{P_s}$, the indices r_1, r_2, and r_3 are associated with individuals from $\mathbf{P_s}$.

The parameters of our j2020 algorithm are presented in Table II. Note, at CEC 2019, in jDE100, F_l and CR_l were two tunable parameters, while this competition does not allow them to be tuned for each benchmark function, i.e., the same parameter value should be used for all benchmark functions.

The j2020 algorithm has not applied the additional mechanism that is used in $\mathbf{P_b}$ only in jDE100. In this mechanism in jDE100 we have a small number $oppNP\%$ of bNP individuals

TABLE VIII
COMPARISON THE MEAN RESULTS FOR 20D.

Func.	DE	jSO	j2020
1	**0.0000**	**0.0000**	**0.0000**
2	254.1268	1.8969	**0.0260**
3	26.8776	20.8927	**14.4196**
4	1.3512	0.5246	**0.1800**
5	28.2255	**12.4058**	77.7693
6	10.7794	1.1336	**0.1915**
7	12.2037	**0.4307**	1.9843
8	100.0000	100.0000	**92.7213**
9	413.6728	400.3541	**339.4512**
10	413.6602	413.6577	**399.0631**

that are trying to solve maximization of function, i.e, which is opposite to default assumption which is minimization for benchmark functions in this paper. After the objective function is calculated, say $c = f(\vec{u})$, then in the selection operation we first check, if the value c is better than the objective value of the $\vec{x}_{best}$ vector. In that case, we need to save $\vec{u}$. Otherwise, vector $\vec{u}$ survives if it is worse (maximization) than its corresponding parent vector. As we mentioned this mechanism is not applied in j2020.

The j2020 uses a crowding mechanism after a crossover operation in order to find an individual which is the closest one the trial vector based on the Euclidian distance. The found individual then compete with the trial individual during a selection operation.

IV. EXPERIMENTS

A. Benchmark Functions

We tested the new j2020 algorithm on ten CEC 2020 special session benchmark functions for single objective bound-constrained optimization [17]. The benchmark functions are collected and presented in Table I. One benchmark function is an unimodal function, three functions belong to a group of basic functions, three of them are hybrid functions, and three of them belong to a group of composition functions. The optimal solution values are known for all benchmark functions and they are presented in the last column of Table I. The upper and lower bounds of a search space are defined as $[-100, 100]^D$, i.e., a lower bound for all components is -100, while an upper bound for all components is 100. The functions are scalable and have dimensions $D = 5, 10, 15$ and 20. The goal is to compute each function's minimum value. The maximum number of objective function evaluations has not been limited in the previous competition (CEC 2019), but this challenge, as several previous competitions, has the limitation of the maximum number of function evaluations. A run for each benchmark function is terminated after $maxFEs$ function evaluations, and they are defined as follows:

- $maxFEs = 50000$ for $D = 5$,
- $maxFEs = 1000000$ for $D = 10$,
- $maxFEs = 3000000$ for $D = 15$, and
- $maxFEs = 10000000$ for $D = 20$.

The organizers of the challenge ask contestants to record the function value ($F_i^* = F_i(\vec{x}^*)$) after $\left\lfloor D^{\frac{k}{5}-3} maxFEs \right\rfloor$ (k=0, 1, 2, 3, ..., 15) for each run. Therefore, 16 error values are recorded for each function for each run. The participants are required to send the final results as the specified format to the organizers who will present an overall analysis and comparison based on these results. Note, error values smaller than 10^{-8} are taken as zero.

For each function, 30 consecutive runs of an algorithm are required. The error values achieved after $maxFEs$ in 30 runs are sorted from the smallest (best) to the largest (worst) and we present the best, worst, mean, median and standard deviation values of function error values for the 30 runs in the next section.

B. Experimental Results

The obtained results of the proposed j2020 algorithm are shown in Tables III–VI.

A comparison on $D = 20$ based on the mean values of the proposed j2020 algorithm with the original DE and jSO algorithms is shown in Table VIII. The best-obtained results for each function of the compared algorithms are presented in **bold**. The original DE uses a standard setting for control parameter $F = 0.5$, $CR = 0.9$, and $NP = 100$ for all functions and all dimensions. The parameter settings for jSO are as in [2]. We can see that algorithms are shown similar performance on function F1. jSO obtained the best results on F5 and F7, while j2020 obtained the best results on F2–F4, F6, F9, and F10.

Execution times are shown in Table VII. Note that we could not compute $(T2 - T1)/T0$ since T0 is zero – compiler has made a simplification during the optimization phase of compilation.

In our experimental work we used the next PC configuration: System: GNU Linux, CPU: Intel(R) Core(TM) i7-4770 CPU 3.4 GHz, Main memory: 16 GB, Programming language: C++, Algorithm: j2020, Compiler: g++ (GNU Compiler).

V. CONCLUSIONS

We proposed a new differential evolution algorithm for solving real parameter single objective bound-constrained optimization. Our algorithm uses self-adaptive control parameters F and CR, two populations with different sizes, restart mechanism in both populations, migration of the best individual from the bigger population into the smaller population, modified mutation strategy in the big population, and crowding mechanism.

In experimental work, our algorithm was applied on ten benchmark functions for $D = 5, 10, 15$, and 20, and results are presented in tables as required by the organizers of the CEC 2020 competition. The obtained results of the proposed j2020 algorithm are compared with the original DE and jSO algorithms for dimension $D = 20$, and j2020 shows better performance.

ACKNOWLEDGMENT

The authors would also like to acknowledge the efforts of the organizers of this session and source code availability of the benchmark functions.

REFERENCES

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

[2] J. Brest, M. S. Maučec, and B. Bošković, "Single objective real-parameter optimization: Algorithm jSO," in *IEEE Congress on Evolutionary Computation (CEC) 2017*. IEEE, 2017, pp. 1311–1318.

[3] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[4] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 27–54, 2011.

[5] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1–2, pp. 61–106, 2010.

[6] 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," Nanyang Technological University, Singapore, Tech. Rep., November 2018. [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[7] A. Kumar, R. K. Misra, D. Singh, and S. Das, "Testing A Multi-Operator based Differential Evolution Algorithm on the 100-Digit Challenge for Single Objective Numerical Optimization," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 34–40.

[8] A. Alić, K. Berkovič, B. Bošković, and J. Brest, "Population Size in Differential Evolution," in *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing*. Springer, 2019, pp. 21–30.

[9] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan, "The 2019 100-Digit Challenge on Real-Parameter, Single Objective Optimization: Analysis of Results." [Online]. Available: https://github.com/P-N-Suganthan/CEC2019

[10] Q. B. Diep, I. Zelinka, S. Das, and R. Senkerik, "SOMA T3A for Solving the 100-Digit Challenge," in *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing*. Springer, 2019, pp. 155–165.

[11] J. Lu, X. Zhou, Y. Ma, M. Wang, J. Wan, and W. Wang, "A Novel Artificial Bee Colony Algorithm with Division of Labor for Solving CEC 2019 100-Digit Challenge Benchmark Problems," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 387–394.

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

[13] T. Eltaeib and A. Mahmood, "Differential Evolution: A Survey and Analysis," *Applied Sciences*, vol. 8, no. 10, p. 1945, 2018.

[14] P. Sharma, H. Sharma, S. Kumar, and J. C. Bansal, "A Review on Scale Factor Strategies in Differential Evolution Algorithm," in *Soft Computing for Problem Solving*. Springer, 2019, pp. 925–943.

[15] M. S. Maučec and J. Brest, "A review of the recent use of Differential Evolution for Large-Scale Global Optimization: An analysis of selected algorithms on the CEC 2013 LSGO benchmark suite," *Swarm and Evolutionary Computation*, vol. 50, p. 100428, 2019.

[16] K. R. Opara and J. Arabas, "Differential Evolution: A survey of theoretical analyses," *Swarm and evolutionary computation*, vol. 44, pp. 546–558, 2019.

[17] 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," Zhengzhou University, Zhengzhou China, and Nanyang Technological University, Singapore, Tech. Rep., November 2019. [Online]. Available: https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark

[18] J. Brest, S. Greiner, B. Bošković, 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, no. 6, pp. 646–657, 2006.

[19] U. Mlakar, B. Potočnik, and J. Brest, "A hybrid differential evolution for optimal multilevel image thresholding," *Expert Systems with Applications*, vol. 65, pp. 221–232, 2016.

[20] B. Bošković and J. Brest, "Protein folding optimization using differential evolution extended with local search and component reinitialization," *Information Sciences*, vol. 454, pp. 178–199, 2018.

[21] A. Zamuda and J. Brest, ""self-adaptive control parameters' randomization frequency and propagations in differential evolution"," *Swarm and Evolutionary Computation*, vol. 25, pp. 72–99, 2015.

[22] R. P. Parouha and K. N. Das, "A memory based differential evolution algorithm for unconstrained optimization," *Applied Soft Computing*, vol. 38, pp. 501–517, 2016.

[23] I. Poikolainen, F. Neri, and F. Caraffini, "Cluster-based population initialization for differential evolution frameworks," *Information Sciences*, vol. 297, pp. 216–235, 2015.

[24] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution, A Practical Approach to Global Optimization*. Springer, 2005.

[25] M. S. Maučec, J. Brest, B. Bošković, and Z. Kačič, "Improved Differential Evolution for Large-Scale Black-Box Optimization," *IEEE Access*, 2018.

[26] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[27] J. Zhang and A. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[28] R. Tanabe and A. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *2014 IEEE Congress on Evolutionary Computation (CEC2014)*. IEEE, 2014, pp. 1658–1665.

[29] J. Yeh, T. Chen, and T. Chiang, "Modified L-SHADE for Single Objective Real-Parameter Optimization," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 381–386.

[30] J. Brest and M. S. Maučec, "Population Size Reduction for the Differential Evolution Algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.