The 100-Digit Challenge: Algorithm jDE100

Janez Brest, Mirjam Sepesy Maučec, Borko Bošković
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ul. 17, 2000 Maribor, Slovenia
Email: janez.brest@um.si, mirjam.sepesy@um.si, borko.boskovic@um.si

***Abstract*—Real parameter optimization problems are often very complex and computationally expensive. We can find such problems in engineering and scientific applications. In this paper, a new algorithm is proposed to tackle the 100-Digit Challenge. There are 10 functions representing 10 optimization problems, and the goal is to compute each function's minimum value to 10 digits of accuracy. There is no limit on either time or the maximum number of function evaluations. The proposed algorithm is based on the self-adaptive differential evolution algorithm jDE. Our algorithm uses two populations and some other mechanisms when tackling the challenge. We provide the score for each function as required by the organizers of this challenge competition.**

***Index Terms*—differential evolution, optimization, global optimum, accuracy**

I. Introduction

100-Digit Challenge [1] on CEC 2019 belongs to single-objective real parameter optimization where the goal is to find a global optimum. Single-objective real parameter optimization plays an important role in various research fields, where increasing efforts are focused on solving complex optimization problems. Several algorithms have been investigated for solving real-parameter problems, and among them, the population based algorithms seem very useful and robust. In this paper, we are dealing with a differential evolution algorithm that belongs to the group of evolutionary algorithms. A quick look to a history of real-parameter single objective optimization competitions and/or special sessions tells us that they were first organized at CEC in 2005, and each year after 2013.

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.

In this paper, we will assume the optimization problems are required to be minimized. However, a maximization problem $max(f(\vec{x}))$ can be transformed into minimization variant as $-min(-f(\vec{x}))$.

A goal in the global optimization problem is to find a 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 define a global optimization problem also as bound-constrained optimization. D denotes the dimensionality of the problem.

In a single objective optimization, function f might have several optima and an algorithm is required to find the global one. If an algorithm traps into a local optimum, the final result of optimization may be poor [2]. Next challenge is regarding the dimensionality of an optimization problem since the search space is becoming very large when the dimension of the problem is increased.

The Differential Evolution (DE) [3] algorithm is a stochastic population-based algorithm. In recent decades, it was very competitive when it was applied to solve real-world optimization problems [4], [5]. DE demonstrates robustness and efficiency, and it is suitable especially for optimization over continuous spaces. However, we can use DE also in domains of discrete optimization.

The original DE was proposed more than 20 years ago. Till now researches used the original algorithm or improved variants in their applications to solve problems in different domains. A huge number of improvements were proposed recently [6]. Further readings include recently published reviews and surveys [7], [8], [9].

In this paper, we present a new version of DE algorithm, called jDE100, for single objective real-parameter optimization. It is based on our previously published algorithm jDE [10]. The main new features of the jDE100 algorithm are as follows. It uses two populations. Lower and upper limits of the control parameters are set differently compared to original jDE. Additionally, it uses a simple one-way migration of currently best individual among the populations. Next, the proposed algorithm applies restart's mechanism separately in both populations to manage a population diversity.

The main contributions of this paper are: (1) A new version of the algorithm (jDE100) with two populations is presented, (2) Experiments for CEC 2019 competition on the 100-Digit Challenge are conducted, (3) The obtained results are presented in a form required by the competition organizers.

The structure of the paper is as follows. Section II gives background for this work, where an overview of the DE and jDE algorithms is presented. Section III proposes our new algorithm, called jDE100, which is used for solving *The 100-Digit Challenge*. In Section IV experimental results of the jDE100 algorithm on problems of the challenge are presented. Section V concludes the paper with some final remarks.

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

II. Background

This section gives some backgrounds on DE and jDE. Differential evolution (DE) is a population-based algorithm that belongs to the group of evolutionary algorithms. It shows highly competitive performance in many applications and researches [11], [12], [13], [14], [15]. The original DE algorithm has three control parameters, F, CR, and NP, that are required to be set by the user before the evolutionary process starts: F is a scaling factor, CR is crossover parameter, and NP is population size. They are fixed in the original DE algorithm, i.e., they have the same value during the whole evolutionary process or optimization task. As it can be observed based on researches, a particular parameter's value can lead to a faster algorithm's convergence rate. If a user needs to set and tune some parameters, it can take a while before a user finds reasonably good values for parameters. To overcome this issue, numerous adaptive and self-adaptive mechanisms for control parameters have been proposed. Control parameters F and CR are self-adaptive in many DE algorithms, like SADE [16], jDE [10], JADE [17], LSHADE [18], jSO [2], etc., and, therefore, a user does not need to tune these two control parameters. However, it needs to set NP parameter. Usually, NP remains fixed during the optimization process. Recently, there have been several attempts to adjust this control parameter during the evolutionary process [19], [18].

A. Differential Evolution

The DE algorithm [3] is a population based algorithm. Its population $\mathbf{P}$ is combined of NP individuals or vectors:

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

where g denotes a generation index, $g = 1, 2, \ldots, G_{MAX}$. DE stops an evolutionary process after G_{MAX} generations. The stopping criteria can be expressed also by the maximum number of function evaluations, time limit, etc. Each vector

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

consists of D variables.

1) Initialization: Before the evolutionary process starts, a population is randomly initialized, i.e., each vector gets randomly generated uniformly distributed values between lower and upper bound for all its components:

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

Then the population of NP vectors is evolved and guides the vectors in a search space toward a global optimum. At the end of the evolutionary process, DE stops and returns the best-fitted vector and its fitness value as the final solution.

During each generation, DE employs three operations for each individual, namely mutation, crossover, and selection.

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 one of the most frequently used mutation strategy in DE. This strategy randomly selects two vectors and their difference is multiplied by scale factor F and added to 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 indexes 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 [20], [21]:

- "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 indexes 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.

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

```
Initialization(P);                                   ▷ population initialization
while stopping criteria is not met do
    for (i ← 0; i < NP; i++) do
        ▷ ** mutation **
        Randomly select r1, r2, and r3;              ▷ r1 ≠ r2 ≠ r3 ≠ i
        v_{i,g+1} ← x_{r1} + F(x_{r2} − x_{r3})
        ▷ ** crossover **
        j_rand ← rand{1, D}
        for (j ← 0; j < D; j++) do
            if (rand(0,1) ≤ CR or j = j_rand) then
                u_{i,j,g+1} ← v_{i,j,g+1}
            else
                u_{i,j,g+1} ← x_{i,j}
            end if
        end for
        ▷ ** selection **
        if (f(u_{i,g+1}) ≤ f(x_{i,g})) then          ▷ minimization
            x_{i,g+1} ← u_{i,g+1}
        else
            x_{i,g+1} ← x_{i,g}
        end if
    end for
end while
```

Algorithm 1: Differential Evolution.

TABLE I
SUMMARY OF THE FUNCTIONS IN THE 100-DIGIT CHALLENGE [1].

No.	Functions	$F_i^* = F_i(x^*)$	D	Search Range
1	Storn's Chebyshev Polynomial Fitting Problem	1	9	[-8192,8192]
2	Inverse Hilbert Matrix Problem	1	16	[-16384,16384]
3	Lennard-Jones Minimum Energy Cluster	1	18	[-4,4]
4	Rastrigin's Function	1	10	[-100,100]
5	Griewank's Function	1	10	[-100,100]
6	Weierstrass Function	1	10	[-100,100]
7	Modified Schwefel's Function	1	10	[-100,100]
8	Expanded Schaffer's F6 Function	1	10	[-100,100]
9	Happy Cat Function	1	10	[-100,100]
10	Ackley's Function	1	10	[-100,100]

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 2: jDE.

Each strategy has a different ability to maintain the population diversity which might increase/decrease 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 former 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 choosing 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 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 known for DE, but rarely applied in other EAs.

B. jDE Algorithm

The jDE algorithm was introduced in 2006 [10]. It uses 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 $F_{i,g+1}$

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

1: Initialize population $\mathbf{P_b} = (\vec{x}_1, \ldots, \vec{x}_{bNP})$ randomly
2: Initialize population $\mathbf{P_s} = (\vec{x}_1, \ldots, \vec{x}_{sNP})$ randomly
3: Initialize $F_i = 0.5$; $CR_i = 0.9$; $(i \in \{1, bNP{+}sNP\})$
4: **while** stopping criteria is not met **do**
5: Check for a reinitialization of $\mathbf{P_b}$
6: Check for a reinitialization of $\mathbf{P_s}$
7: **for each** $i \in \mathbf{P_b}$ **do**
▷ ** on big population **
8: Perform **jDE mutation** (Steps 5–7 in Alg. 2)
9: Perform **jDE crossover** (Steps 8–16 in Alg. 2)
10: Perform **jDE selection** (Steps 17–25 in Alg. 2)
11: **end for**
12: **if** $\vec{x}_{best} \in \mathbf{P_b}$ **then**
13: Copy $\vec{x}_{best}$ into $\mathbf{P_s}$
14: **end if**
15: **for** $k \in 1, 2, ..., m$ **do** ▷ repeat m-times
16: **for each** $i \in \mathbf{P_s}$ **do**
▷ ** on small population **
17: Perform **jDE mutation** (Steps 5–7 in Alg. 2)
18: Perform **jDE crossover** (Steps 8–16 in Alg. 2)
19: Perform **jDE selection** (Steps 17–25 in Alg. 2)
20: **end for**
21: **end for**
22: **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: jDE100.

and $CR_{i,g+1}$ are calculated before the mutation operation is performed as follows [10]:

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

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

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. 2.

In [10] 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 $[0, 1]$.

In [21] lower and upper bounds of F and CR are used differently compared to the suggestions in [10]. LSGOjDE was proposed in [21] where three mutation strategies are applied. The upper limit for F is set to 1 only for jDE/current-to-best/1 strategy, for jDE/rand/1 strategy it is set to 2. Permitting larger values of F enables the algorithm to make a larger move of the mutant vector $\vec{v}_{i,g+1}$ away from $\vec{x}_{r_1,g}$. Parameter F is not used in the third strategy, i.e., local search, as this strategy is not subjected to mutation.

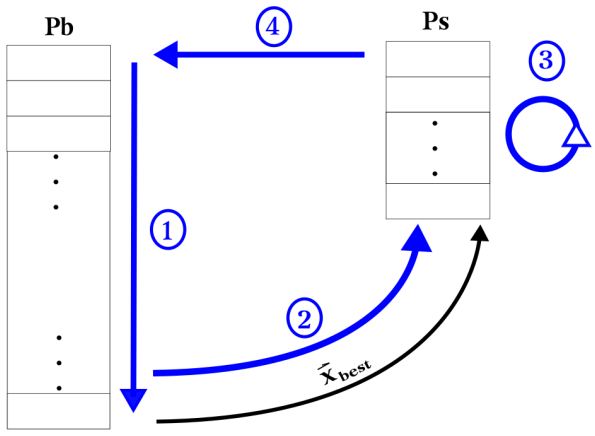

Fig. 1. Evolutionary process in the jDE100 algorithm. Four steps are repeating: ① One generation is performed on big population $\mathbf{P_b}$. ② If required, $\vec{x}_{best}$ is copied into small population $\mathbf{P_s}$. ③ More generations are performed on $\mathbf{P_s}$. ④ The evolutionary process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

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

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

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 [22], a narrower interval is used for self-adapting values of the F control parameter: $[0.01, 0.1]$.

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 $\vec{x}_{i,g+1}$.

III. Algorithm jDE100

In this section, a new algorithm (jDE100) for solving single-objective real parameter optimization is presented and its pseudo-code is given in Algorithm 3. It is an extended version of the jDE [10] algorithm. The new algorithm uses two populations: 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 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 a combination of two conditions:

- a run is terminated when a 10-digit level of accuracy is reached, or

TABLE II
PARAMETERS IN THE JDE100 ALGORITHM. F_l AND CR_l ARE TWO TUNABLE PARAMETERS.

Parameter	Value	Description
F_l	$\frac{5.0}{\sqrt{bNP}}$ for F1–F8, F10, big pop.	lower limit of scale factor for the big population
F_l	$\frac{0.01}{\sqrt{bNP}}$ for F9, big pop.	lower limit of scale factor for the big population
F_l	$\frac{1.0}{\sqrt{sNP}}$ for F1–F8, F10, small pop.	lower limit of scale factor for the small population
F_l	$\frac{0.01}{\sqrt{sNP}}$ for F9, small pop.	lower limit of scale factor for the small population
F_u	1.1	upper limit of scale factor
CR_l	0.0 for F1–F8, F10	lower limit of crossover parameter
CR_l	1.0 for F9	lower limit of crossover parameter
CR_u	1.1	upper limit of crossover parameter
F_{init}	0.5	initial value of scale factor
CR_{init}	0.5	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
bNP	1000	size of $\mathbf{P_b}$
sNP	25	size of $\mathbf{P_s}$
$ageLmt$	$1e9$	number of FEs when population restart needs to occurs
eps	$1e-16$	small value used to check if two value are similar
$maxFEs$	$1e12$	one of stopping condition (max. number of function evaluations). (Actually not used.)
$myEqs$	25	reinitialization if $myEqs\%$ of individuals in the corresponding population have the similar function values

- a run is terminated after $maxFEs = 1e12$.

From the obtained results which are presented in the next section, the latter condition of stopping criteria has never occurred during experiments by the jDE100 algorithm.

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 7–11 in Alg. 3). Mutant vector $\vec{v}_i$ is generated with the jDE/rand/1 strategy. Note, r_1 is the index of individual from $\mathbf{P_b}$, while r_2 and r_3 are indexes of individuals from $\mathbf{P_b} \cup \vec{x}_s$, where $\vec{x}_s$ is one individual from the small population $\mathbf{P_s}$. The motivation of using the proposed ranges for the indexes r_2 and r_3 is to have a small influence of $\mathbf{P_s}$ on $\mathbf{P_b}$. 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.

The mutation, crossover and selection operations on $\mathbf{P_s}$ are similar to those on $\mathbf{P_b}$. They are repeated m-times, which ensures that both populations have an equal amount of function evaluations. Here, we assumed that $bNP = m \times sNP, m \in 1, 2, ...$ Note, in the mutation, the r_1, r_2, and r_3 are indexes of individuals from $\mathbf{P_s}$.

TABLE III
FIFTY RUNS FOR EACH FUNCTION SORTED BY THE NUMBER OF CORRECT DIGITS.

Function	Number of correct digits											Score
	0	1	2	3	4	5	6	7	8	9	10	
F1	0	0	0	0	0	0	0	0	0	0	50	10
F2	0	0	0	0	0	0	0	0	0	0	50	10
F3	0	0	0	0	0	0	0	0	0	0	50	10
F4	0	0	0	0	0	0	0	0	0	0	50	10
F5	0	0	0	0	0	0	0	0	0	0	50	10
F6	0	0	0	0	0	0	0	0	0	0	50	10
F7	0	0	0	0	0	0	0	0	0	0	50	10
F8	0	0	0	0	0	0	0	0	0	0	50	10
F9	0	0	0	0	0	0	0	0	0	0	50	10
F10	0	0	0	0	0	0	0	0	0	0	50	10
											Total:	**100**

TABLE IV
BEST, WORST, MEDIAN, MEAN AND STANDARD DEVIATION VALUES OF THE NUMBER OF FUNCTION EVALUATION (FES) AFTER 10-DIGITS ACCURACY FOR BEST 25 RUNS OUT OF ALL 50 RUNS, AS IT IS REQUIRED BY THE ORGANIZER OF THIS COMPETITION.

No.	best	worst	median	avg	stdDev
1	3.378e+04	6.237e+05	9.53e+04	1.59e+05	1.597e+05
2	2.331e+06	2.431e+06	2.388e+06	2.385e+06	2.719e+04
3	1.377e+05	2.814e+06	1.129e+06	1.31e+06	8.519e+05
4	1.25e+05	5.099e+05	3.718e+05	3.475e+05	1.149e+05
5	5.52e+04	3.771e+05	1.473e+05	1.673e+05	8.426e+04
6	3.501e+04	4.155e+04	3.789e+04	3.841e+04	2.063e+03
7	2.482e+06	1.589e+07	9.185e+06	9.105e+06	4.528e+06
8	1.591e+08	1.779e+09	1.372e+09	1.219e+09	4.388e+08
9	6.086e+08	1.056e+09	9.494e+08	9.207e+08	1.131e+08
10	1.796e+05	2.533e+06	1.761e+06	1.541e+06	7.46e+05

The parameters of our jDE100 algorithm are presented in Table II.

IV. EXPERIMENTS

The jDE100 algorithm was tested on ten CEC 2019 special session benchmark functions of the *100-Digit Challenge* [1] and the functions are collected and presented in Table I. The functions have dimensions from 9 to 18, but most of them have $D = 10$. In the last column of Table I, the upper and lower bounds of a search space are shown. The optimal solution values are known for all benchmark functions in this challenge. The minimum for all functions to ten digits of accuracy is 1.000000000. The goal is to compute each function's minimum value to 10 digits of accuracy. The maximum number of objective function evaluations has been limited in previous competitions, but this challenge has no limit on either time or the maximum number of function evaluations.

A. Evaluation Method

In the 100-Digit Challenge [1] there are 10 problems, which in our case are 10 functions, and the goal is to compute each function's minimum value to 10 digits of accuracy without being limited by time. The challenge asks contestants to solve all ten problems with one algorithm, although limited control parameter tuning for each function is permitted to restore some of the original contests flexibility.

For each function, 50 consecutive runs of an algorithm are required, each with a different initial population. The total number of correct digits in the 25 runs that have the lowest function values are collected. The score for that function is the average number of correct digits in the best 25 trials, i.e. if 50% or more of the trials find all 10 digits, then the score for that function is a perfect 10. The maximum score for the ten function total is 100, i.e. when the best 25 out of 50 trials for all 10 functions give the minimum to 10-digit accuracy.

The organizers of the challenge ask contestants to record the number of function evaluations (FEs) after (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) digit(s) of accuracy for each of the best 25 runs. The participants are required to send the final results as by the specified format to the organizers who will present an overall analysis and comparison based on these results.

B. Experimental Results

The obtained score results of the proposed algorithm, named jDE100, are presented in Table III. This table lists the number of trials for each function in a run of 50 that found n correct digits, where $n = 1, 2, ..., 10$. In the final column, the average number of correct digits in the best 25 runs is shown, i.e., the score for that function. The total score is calculated as the sum of the scores for all 10 functions, and it is presented in the last column. Our algorithm has obtained a perfect score for all functions in the challenge (F1–F10), and the total score is **100**.

The competition rules regarding parameters and operators are as follow. One may tune up to 2 parameter(s) and/or

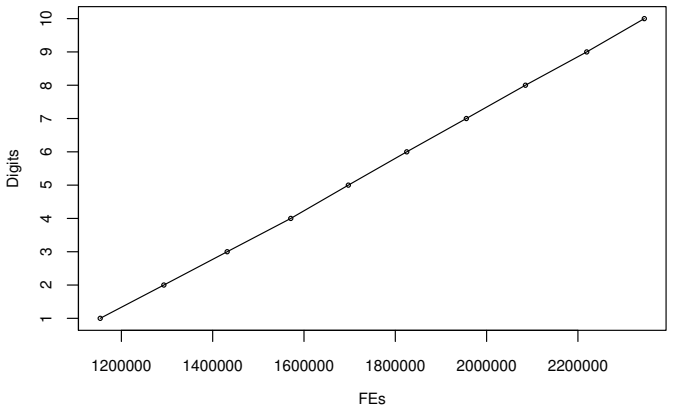

Fig. 2. Convergence graph for function F2.

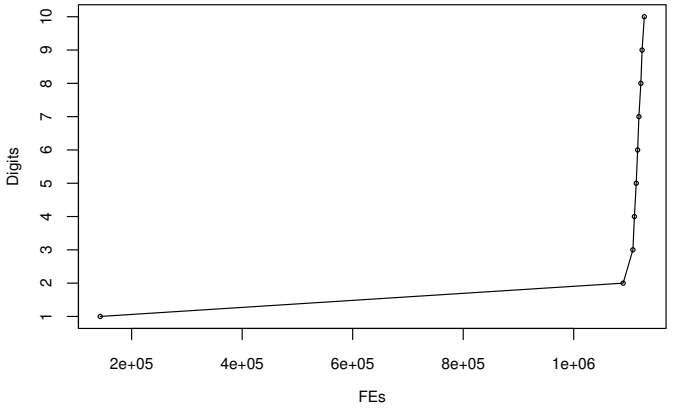

Fig. 3. Convergence graph for function F3.

operator(s) independently for each problem. There is no limit to the number of parameters and operators that are fixed or adjusted in the identical manner for all 10 problems, but the two tunable parameter(s) and/or operator(s) must be the same for all 10 problems. Adaptive parameters do not count as tuned parameters provided that they are both initialized and adapted in the identical manner for all 10 problems. Table II shows the values of parameters in our algorithm.

Table IV presents best, worst, median, mean and standard deviation values of the number of function evaluation (FEs) after 10-digits accuracy for each function of the best 25 runs. Since mean (avg) values are similar to standard deviation values we can assume that the distribution of the FEs is near-exponential or near-geometric.

Convergence graphs – the number of function evaluations required to get digits accuracy – for functions F2, F3, and F8 are presented on Figures 2, 3, and 4, respectively. One can see a linear dependency of digits accuracy and number of function evaluations on function F2. The algorithm required the biggest amount of FEs to get 6 digit accuracy on function F3, while on function F8 the biggest amount of FEs were needed from 2 digits to get to 3 digits accuracy.

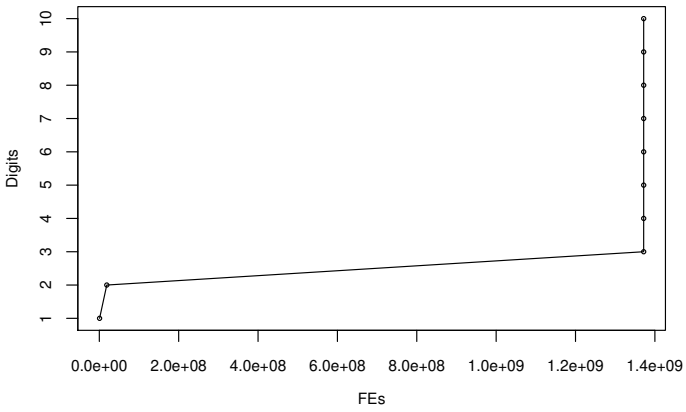

Fig. 4. Convergence graph for function F8.

PC configuration:
System: GNU Linux, CPU: Intel(R) Core(TM) i7-4770 CPU 3.4 GHz, Main memory: 16 GB, Programming language: C++, Algorithm: jDE100, Compiler: g++ (GNU Compiler).

V. Conclusions

In this paper, we proposed a new algorithm for solving the problems of *The 100-Digit Challenge*. Our algorithm uses self-adaptive control parameters F and CR, two populations, migration of the best individual from the bigger population into the smaller population, restart mechanism in both populations.

The proposed algorithm obtained the scores 10 (the perfect) for functions F1–F10, i.e. for all functions in this challenge. The total score is 100, which is the highest possible score value.

Acknowledgment

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

References

[1] 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/

[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] 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.

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

[8] 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.

[9] 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*, 2018.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] J. Brest, A. Zamuda, I. Fister, and M. S. Maučec, "Large scale global optimization using self-adaptive differential evolution algorithm," in *2010 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010, pp. 1–8.