Population Size in Differential Evolution*

Amina Alić[1], Klemen Berkovič[1], Borko Bošković[1][0000−0002−7595−2845], and Janez Brest[1][0000−0001−5864−3533]

Computer Architecture and Languages Laboratory, Institute of Computer Science, Faculty of Electrical Engineering and Computer Science, University of Maribor, SI-2000 Maribor, Slovenia
amina.alic@um.si, klemen.berkovic1@um.si, borko.boskovic@um.si, janez.brest@um.si
https://labraj.feri.um.si

Abstract. In this paper we examined how the population size affects the performance of the differential evolution algorithm. First, we tested the original differential evolution algorithm, and then the improved self-adaptive differential evolution algorithm, on ten benchmark functions, that have been proposed for the CEC 2019 competition. We used six different population sizes. Afterwards, we tested the newly created algorithm with population reinitialization. The results show that the population size affects the algorithm's efficiency, and that we need to tune it to obtain the best results. In the paper, we demonstrate that the newly created algorithm with reinitialization gives better, or at least comparable, results than the two algorithms without reinitialization.

Keywords: Global optimum · Differential evolution · Reinitialization · Population size.

1 Introduction

Evolutionary computing is a research area inspired by natural evolution [3]. The main feature of natural evolution is the survival of the fittest. In evolutionary algorithms, the initial population is generated randomly and the fitness of every individual is calculated. The best ones survive and reproduce, and so evolution progresses [3]. Because of simplicity, in evolutionary algorithms, the population size NP is constant, but we are aware that this is not the case in nature. Instead, the number of individuals in a population varies in different generations. Adaptive population size is still a challenging task, and, for now, we wanted to see how the population size affects the algorithms.

We briefly discus the related work in Section 2. The original differential evolution algorithm and its improved self-adaptive version are described in Section 3.

* The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0041), and the investment co-financed by the Republic of Slovenia and the European Union, European Regional Development Fund, implemented under the Operational Program for the Implementation of the EU Cohesion Policy in the period 2014–2020.

A new algorithm is proposed in Section 4. Section 5 presents our experiments and results. In Section 6 we conclude our work briefly.

2 Related work

In the paper [6], the authors investigated how different population sizes affect the differential evolution algorithm. The paper considered the effect of the population sizes 2D, 4D, 6D, 8D and 10D (D - dimension of the problem) with two different mutation strategies on problems chosen from the CEC 2005 Special Session on Real-Parameter Optimization. They found that a smaller population size with a greedy strategy converges fast but premature convergence and stagnation are more pronounced. A large population, with a strategy having good exploration capacity, does not prematurely converge or stagnate but it can converge very slow.

The same authors in another paper [5] have proposed a differential evolution algorithm with an ensemble of parallel populations having different population sizes, in which a more suitable population size takes most of the function evaluations adaptively. Although this paper uses multiple populations, it is related to our work in the manner that it explores how the population size affects the convergence. They found that the multi-population differential evolution algorithm was more efficient in obtaining better quality solutions than the conventional differential evolution algorithm.

A review article on the study of how the population size affects differential evolution [9] emphasizes that the inappropriate choice of the population size may seriously impede the performance of each differential evolution algorithm.

All those papers are considering the adaptation of population size in the conventional differential evolution algorithm, with other parameters (crossover and mutation rates) kept constant. Besides that, we are investigating the effect of changing population size along with adapting other parameters.

3 Background

3.1 Differential Evolution (DE)

In the DE algorithm [12, 11, 2, 4, 8, 7], population or candidate solutions are represented by real-valued vectors with D components (genes) [3]

$$\boldsymbol{x}_i^{(G)} = x_{i,1}^{(G)}, x_{i,2}^{(G)}, ..., x_{i,D}{}^{(G)}, \tag{1}$$

where $i = 1, ..., NP$ and NP is the population size. G represents a generation. The offspring are created through the mutation and crossover. The initial population is generated randomly between lower and upper bounds, which are defined by the problem. An evolutionary cycle starts with random selection of 3 vectors

$\boldsymbol{x}_{r1}, \boldsymbol{x}_{r2}, \boldsymbol{x}_{r3}$ from the initial population. A mutation vector is then obtained by adding a perturbation vector to the first of those random vectors

$$\boldsymbol{v}_i^{(G+1)} = \boldsymbol{x}_{r1}^{(G)} + \boldsymbol{p}^{(G)}. \tag{2}$$

The perturbation vector $\boldsymbol{p}$ is the difference between two other randomly selected vectors multiplied by the scaling factor F

$$\boldsymbol{p}^{(G)} = F(\boldsymbol{x}_{r2}^{(G)} - \boldsymbol{x}_{r3}^{(G)}). \tag{3}$$

The scaling factor is a positive number, and has values $F \in [0, \infty]$. In most cases, the scaling factor occupies values between $F \in [0, 2]$. The second step in the reproduction is usually a binomial crossover, which has one parameter, the crossover probability $CR \in [0, 1]$. It creates a trial vector, combining elements of the mutation vector and the corresponding parent vector as

$$u_{i,j}^{(G+1)} = \begin{cases} v_{i,j}^{(G+1)}, & \text{if } rand(0,1) \leq CR \text{ or } j == j_{rand}, \\ x_{i,j}^{(G)}, & \text{otherwise.} \end{cases} \tag{4}$$

CR determines the probability that the trial vector takes a component from the mutation vector, and j_{rand} is a randomly chosen integer in the range $1, ..., D$ which provides that at least one component of the trial vector is changed in regard to the previous generation. A selection of the offspring that will proceed to reproduction in next generation comes at the end of the evolutionary cycle. The fitness value of the trial vector is compared to the fitness value of the previous population member, the parent vector. The fittest member is allowed to be reproduced further

$$\boldsymbol{x}_i^{(G+1)} = \begin{cases} \boldsymbol{u}_i^{(G+1)}, & \text{if } f(\boldsymbol{u}_i^{(G+1)}) \leq f(\boldsymbol{x}_i^{(G)}), \\ \boldsymbol{x}_i^{(G)}, & \text{otherwise.} \end{cases} \tag{5}$$

3.2 Self-Adaptive Differential Evolution (jDE)

DE has three parameters, namely F, CR, and NP, and their tuning can improve the performance of the algorithm greatly. In the original DE these parameters are specified before the evolutionary cycle, and remain fixed during each generation of the algorithm. That is in contrast with the dynamic nature of evolutionary computing itself. Furthermore, different values of parameters can be optimal at different stages of the evolutionary process. A better approach is to use self-adapting parameters. In an improved algorithm, that the authors named jDE [1], all population members are extended by the control parameters F and CR. Adaptive changes of the control parameters should give better individuals, in the sense that they will have better fitness values. In jDE new control parameters are calculated as

$$F_i^{(G+1)} = \begin{cases} F_l + rand_1 * F_u, & \text{if } rand_2 < \tau_1 \\ F_i^{(G)}, & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 1: Self-Adaptive Differential evolution with Restarts

Input: NP, F_l, F_u, τ_1, τ_2, α, ε
Output: Best found solution

```
Initialization;
while not stopping criteria met do
    count ← 0;
    for i ← 1 to NP do
        Use Eq. (6) for obtaining new value of F_i;
        Use Eq. (7) for obtaining new value of CR_i;
        Use Eq. (2) for creating new mutant vector v_i, where F = F_i and
          CR = CR_i;
        for j ← 1 to D do
            Use Eq. (4) for crossing over component j;
        end
        Use Eq. (5) for selection between individual x_i^(G) and x_i^(G+1) ;
        if f(x_b) < f(x_i^(G+1)) then
            x_b ← x_i^(G+1);
        end
        if |f(x_i^(G+1)) − f(x_b)| < ε then
            count ← count + 1;
        end
    end
    if count ≥ NP · α then
        Population reinitialization;
    end
end
return x_b;
```

and

$$CR_i^{(G+1)} = \begin{cases} rand_3, & \text{if } rand_4 < \tau_2 \\ CR_i^{(G)}, & \text{otherwise.} \end{cases} \tag{7}$$

Here, τ_1 and τ_2 represent small probabilities when control parameters should be changed, and $rand_j$, $j = (1, 2, 3, 4)$ are uniform random numbers in the range $[0, 1]$. $F_{i,G+1}$ and $CR_{i,G+1}$ are computed before the mutation, so they have an impact on mutation, crossover and selection operations when making offspring.

4 Self-Adaptive Differential Evolution with Restarts (rjDE)

Our main work is based on the newly created algorithm, called rjDE, that is presented in Alg. 1. The algorithm is designed for tackling CEC 2019 problems proposed in the technical report [10] for the 100-Digit challenge. The proposed rjDE algorithm is derived from the jDE algorithm, so it uses the same technique for control parameters adaptation over each evolutionary step.

The jDE algorithm can have the same problem as the DE algorithm. Both algorithms have fast convergence, and can be trapped into local optima. When

Table 1. Fifty runs of DE 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	4	42	0	1	0	0	0	0	0	0	3	2.16
F4	48	2	0	0	0	0	0	0	0	0	0	0.08
F5	0	0	35	9	0	0	0	0	0	0	6	4.28
F6	1	34	0	0	0	0	0	0	0	0	15	6.4
F7	49	1	0	0	0	0	0	0	0	0	0	0.04
F8	35	15	0	0	0	0	0	0	0	0	0	0.6
F9	0	18	32	0	0	0	0	0	0	0	0	2
F10	50	0	0	0	0	0	0	0	0	0	0	0
											Total:	**35.56**

the basic DE converges to some local optima, its population diversity is decreased. In order to avoid that, we added Line 15 to Alg. 1 that checks if the i-th individual fitness value is close to that of the best individual. If this condition is true, we increase the counter. When some individuals obtained similar fitness values as the best one, we can assume that population diversity is also decreased. Therefore, a reinitialization of the population takes place. A restart in rjDE , i.e. population reinitialization, occurs when the fitness values of $\alpha \cdot NP$ individuals differ from the best fitness value by less than a very small value ε.

5 Experiments and Results

We experimented on CEC benchmark functions [10] and followed their rules for computing scores. In [10] there is no limit on the maximum number of function evaluations ($MaxFEs$), but in this work we set $MaxFEs = 10^7$.

We analyzed three different algorithms: DE, jDE, and rjDE on population sizes $NP = 50, 100, 200, 400, 800, 1600$.

We show results for population size 100 first, and, later, compare results for all population sizes, on all 10 functions, including all 3 algorithms.

5.1 Results for $NP = 100$

Table 1 shows the results for all benchmark functions using the original DE algorithm. Other parameters that we used in this algorithm are $F = 0.5$ and $CR = 0.9$. It can be seen that DE obtained 10 correct digits for functions F1 and F2, while for function F10 it obtained no correct digits. Functions F4, F7 and F8 also have scores almost equal to zero, from which we can see that the original DE algorithm is not suitable for solving those functions when using a population with size 100.

The jDE algorithm solved functions F1, F2 and F4 to 10 correct digits. For all other functions, scores were bigger than one correct digit. Those results are

Table 2. Fifty runs of jDE 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	1	49	10
F3	0	39	0	0	0	0	0	0	0	0	11	4.96
F4	2	2	0	0	0	0	0	0	0	0	46	10
F5	0	17	12	0	0	0	0	0	0	0	21	8.88
F6	0	31	0	0	0	0	0	0	0	0	19	7.84
F7	13	31	6	0	0	0	0	0	0	0	0	1.24
F8	0	47	3	0	0	0	0	0	0	0	0	1.12
F9	0	0	49	1	0	0	0	0	0	0	0	2.04
F10	36	0	0	0	0	0	0	0	0	0	14	5.6
											Total:	**61.68**

Table 3. Fifty runs of rjDE 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	2	24	19	0	0	1	0	0	4	5.04
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	2	3	1	0	0	0	0	0	0	0	44	10
F8	0	48	2	0	0	0	0	0	0	0	0	1.08
F9	0	0	50	0	0	0	0	0	0	0	0	2
F10	0	0	0	0	0	0	0	0	0	0	50	10
											Total:	**78.12**

shown in Table 2. Parameters used in this algorithm are $F_l = 0.1$, $F_u = 0.9$, $\tau_1 = \tau_2 = 0.1$. Initial control parameters were $F = 0.5$ and $CR = 0.9$. It is obvious that jDE is better for solving single objective optimization problems than the simple DE.

Table 3 shows the score for each function for the rjDE algorithm. The highest score 10 was obtained for 7 out of 10 functions. Parameters used here were the same as for the jDE algorithm, along with two additional parameters, $\varepsilon = 10^{-16}$ and $\alpha = 0.5$. Functions F3, F8, and F9 seem to be the difficult ones for rjDE.

Total scores for DE, jDE and rjDE are 35.56, 61.68 and 78.12, respectively.

5.2 Results for Different Population Sizes

In Table 4 we present the performance of the DE algorithm on all 10 benchmark functions and all 6 population sizes. It is obvious that performance of DE is not increasing nor decreasing continuously when the population size increases,

Table 4. Scores for each function and each population size using DE.

Function	Population size					
	50	100	200	400	800	1600
F1	10	10	10	10	10	10
F2	2.04	10	10	10	10	6
F3	3.52	2.16	1	1.08	1.28	1.52
F4	0.08	0.08	0.52	2.08	0	0
F5	3.16	4.28	3.36	6.04	2.44	1
F6	5.32	6.4	6.76	7.48	10	10
F7	0	0.04	0	0	0.12	0.44
F8	0.2	0.6	1.04	1.2	0.28	0
F9	1.76	2	2	2	2	2
F10	0	0	0.4	2.8	1.56	0
Total	36.08	35.56	35.08	**42.68**	37.68	30.96

but we can observe that it has by far the best score for $NP = 400$. Observing all particular functions, it can be seen that the best performance was for the function F1, namely 10 correct digits were obtained for every population size. For the function F2, DE obtained 10 correct digits for population sizes $NP = 100, 200, 400, 800$. Too small and too big population sizes obviously have a bad impact on this function, but too small an NP is still worse than too big. For function F6, the DE algorithm reached 10 correct digits for $NP = 800, 1600$, and all other scores were bigger than 5. Functions F3, F5 and F9 all have scores equal to or greater than 1, while for the remaining functions, zero correct digits were obtained 2 or 3 times. The best total score, equal to 42.68, was obtained for $NP = 400$.

Performance of the jDE algorithm is shown in Table 5. It is obvious that this algorithm has better performance than the previous one. The best score (10 correct digits) was obtained 17 times out of 60 possibilities. For function F1 jDE obtained 10 correct digits for every population size, for F2 and F5 4 times and for F4 3 times. Zero correct digits were obtained only 4 times. The algorithm performed worst on function F7. The best total score, 66.48, was obtained again for population size 400.

Table 6 presents results for the rjDE algorithm. This algorithm reached 10 correct digits 37 times out of 70, which is 52.85% of overall performance. The worst result, zero correct digits, was reached only once, for function F10 and population size 1600. The best total score, 78.6, was obtained for population size of 50.

It is obvious that the population size affects the performance of all differential evolution algorithms. For the jDE and rjDE it seems that the total score increases when we increase the population size until it reaches the maximum for some NP, and then decreases for bigger population sizes. For the DE algorithm, we can notice a small deviation from that observation. The maximum for different algorithms is obtained for different population sizes: For DE the best population size is 400, followed by 800, for jDE the best population sizes are

Table 5. Scores for each function and each population size using jDE.

Function	Population size					
	50	100	200	400	800	1600
F1	10	10	10	10	10	10
F2	1.92	10	10	10	10	10
F3	2.08	4.96	6.4	7.72	10	0.2
F4	6.76	10	10	10	1.44	0.08
F5	6.92	8.88	10	10	10	10
F6	8.92	7.84	7.12	8.56	9.28	10
F7	0.8	1.24	1.88	0	0	0
F8	1.12	1.12	1	1	0.96	0.2
F9	2.04	2.04	2	2	2	2
F10	5.6	5.6	8	7.2	2.76	0
Total	46.16	61.68	66.40	**66.48**	56.44	42.48

Table 6. Scores for each function and each population size using rjDE.

Function	Population size						
	25	50	100	200	400	800	1600
F1	10	10	10	10	10	10	10
F2	0.04	5.08	10	10	10	10	9.36
F3	10	10	5.04	3	2.48	2	1.2
F4	10	10	10	10	10	4.88	2.2
F5	10	10	10	10	10	10	5.96
F6	10	10	10	10	10	10	10
F7	5.88	10	10	8.6	1	0.64	0.04
F8	1.2	1.52	1.08	1	1	1	1
F9	2	2	2	2	2	2	2
F10	10	10	10	10	10	4.16	0
Total	69.12	**78.6**	78.12	74.6	66.48	54.68	41.76

400, 200, and for rjDE 50 and 100. Obviously, some algorithms perform better on bigger populations, while the others give better results for smaller population sizes. We followed the rules that were suggested for the CEC 2019 competition.

6 Conclusion

We analyzed three algorithms: DE, jDE and rjDE on the CEC 2019 benchmark functions with different population sizes, in order to see how the population size affects their performance. We followed the rules of the CEC 2019 competition. Our analysis shows that the population size affects the performance of those algorithms in the manner that it increases the total score until it reaches maximum. Further increment of the population size decreases the total score. The self-adaptive differential evolution with reinitialization has proven to have the best results when performing on selected benchmark functions. For the future work, we plan to run the algorithms with a greater maximum number of func-

tion evaluations. We also plan to investigate linear population reduction methods such as L-SHADE [13].

References

1. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Transactions on Evolutionary Computation **10**(6), 646–657 (2006)
2. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolution–an updated survey. Swarm and Evolutionary Computation **27**, 1–30 (2016)
3. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing, Springer-Verlag, Berlin (2003)
4. Eltaeib, T., Mahmood, A.: Differential Evolution: A Survey and Analysis. Applied Sciences **8**(10), 1945 (2018)
5. Mallipeddi, R., Suganthan, P.: Differential evolution algorithm with ensemble of populations for global numerical optimization. Opsearch **46**(2), 184–213 (2009)
6. Mallipeddi, R., Suganthan, P.N.: Empirical study on the effect of population size on differential evolution algorithm. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence). pp. 3663–3670. IEEE (2008)
7. Maučec, M.S., Brest, J.: 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 (On line, 2018). https://doi.org/10.1016/j.swevo.2018.08.005
8. Neri, F., Tirronen, V.: Recent advances in differential evolution: a survey and experimental analysis. Artificial Intelligence Review **33**(1–2), 61–106 (2010)
9. Piotrowski, A.P.: Review of differential evolution population size. Swarm and Evolutionary Computation **32**, 1–24 (2017). https://doi.org/10.1016/j.swevo.2016.05.003
10. Price, K.V., Awad, N.H., Ali, M.Z., Suganthan, P.N.: Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization. Tech. rep., Nanyang Technological University, Singapore (November 2018), http://www.ntu.edu.sg/home/epnsugan/
11. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation **13**(2), 398–417 (2009)
12. Storn, R., Price, K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization **11**, 341–359 (1997)
13. Tanabe, R., Fukunaga, A.S.: Improving the search performance of shade using linear population size reduction. 2014 IEEE Congress on Evolutionary Computation (CEC) pp. 1658–1665 (2014)