Development and analysis of a self-configuring differential evolution algorithm

Zakhar Novikov[1], and *Aleksei* Vakhnin[1,2*]

[1]Siberian Federal University, Institute of Space and Information Technology, Krasnoyarsk, Russia
[2]Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarsk, Russia

Abstract. In this study to solve optimization problem, three differential evolution algorithms are tested on various functions, highlighting its parameter sensitivity. To overcome this, a self-configuring algorithm is introduced, which core idea is to periodically reevaluate configurations, favoring those with superior performance. Self-configuring algorithms in most cases outperform or match conventional methods, enhancing the likelihood of achieving superior results.

1 Introduction

In the field of computational optimization, there exist a class of global optimization problems known as black-box problems. These problems involve finding the optimal solution within a complex and often non-linear search space, where the underlying mathematical functions are either unknown or not readily interpretable [1]. Addressing such challenges is a formidable task that has significant implications in various domains, including engineering, finance and machine learning.

We can define black-box optimization problem as follows: let $f(x_1, x_2, \ldots, x_n)$ be an objective function that maps a vector of input parameters $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to real number $f(x) \in R$. The goal is to find a vector of input parameters $\mathbf{x}^*$ that minimizes (or maximizes) the objective function:

$$f(\boldsymbol{x}^*) = min_x f(\boldsymbol{x}) \quad (1)$$

In practical optimization scenarios, it is often the case that we cannot precisely determine the global minimum (or maximum) of a function due to various factors, including measurement noise or numerical approximations. To account for these uncertainties, the equality (1) is frequently transformed into an inequality:

$$f(\boldsymbol{x}^*) - \varepsilon \leq min_x f(\boldsymbol{x}) \quad (2)$$

Where ε represents an error or tolerance level. Usually the error is set depending on range of objective space and on dimensionality. In this study for all of the experiments we set $\varepsilon = 0.01$.

Black-box global optimization poses a formidable challenge due to the complex and often inscrutable nature of the objective function. Unlike conventional optimization techniques that may rely on specific problem structures, evolutionary algorithms, as metaheuristics, provide

* Corresponding author: alexeyvah@gmail.com

a high-level strategy for navigating these intricate solution spaces. Their effectiveness stems from their ability to perform global search, explore diverse regions of the solution space, and adapt to the uncertainties and complexities inherent in the objective function.

Among the types of evolutionary algorithms often used for this purpose, one of the frequently used ones is the differential evolution (DE). Differential evolution is a prominent evolutionary algorithm widely employed for solving optimization problems, especially those characterized by black-box objective functions. DE is known for its effectiveness in exploring complex and non-linear solution spaces, making it a valuable tool. It should be noted that DE is only applicable to real-valued inputs and it doesn't required a gradient of the optimized function. This algorithm's strength lies in its capacity to navigate diverse landscapes, adapt to noisy or uncertain objective functions, and efficiently discover optimal or near-optimal solutions [2, 3].

DE operates on a population of candidate solutions, employing a set of strategies to create new individuals through mutation, crossover, and selection operations. The algorithm iteratively refines these individuals while aiming to improve the objective function value. One of DE's distinctive features is its ability to balance exploration (searching for diverse regions) and exploitation (converging towards the best-known solution). Algorithm 1 shows concrete implementation of differential evolution algorithm with mutation strategy DE/rand/1, and parameters: weighting coefficient (F) for mutation, the crossover rate (CR) and population size (NP). The algorithm stops when the maximum number of populations (M) is reached, which is set during initialization.

Algorithm 1: Differential evolution

```
// Initialization
Set maximum number of populations (M);
Form start population I_m in form of vectors [x^1, x^2, ..., x^NP];
Set number of population m = 0;
// Main loop
while m < M do
    I_c = [];
    for i in I_m do
        x^t = x^i;
        // Form mutant based on mutation strategy. Here it is DE/rand/1
        x^a, x^b, x^c = select randomly from I_m different to x^i;
        x^ci = x^c + F(x^a – x^b) ;
        if x_s^ci ∉ [a_si, b_si] then x_si^ci = rand from [a_si, b_si] ;
        for j = 1 to length(x^ci) do
            U_i = rand from [0, 1] ;
            if Ui ≤ CR then
                x_i^s = x_i^ci;
            else
                x_i^s = x_i^t;
            end
        end
        if f(x^s) < f(x^t) then
            add x^s to I_c;
        else
            add x^t to I_c;
        end
    end
```

```
    I_m = I_c
    m += 1
end
Select best x from I_c
```

While DE is a powerful optimization tool, the careful configuration of its parameters is crucial to its performance. Parameters such as the weighting coefficient (F) for mutation, the crossover rate (CR), and the mutation strategy [4] play a pivotal role in determining its success in various problem domains. Proper parameter tuning is often a challenging task, as their effectiveness can vary from one problem to another. In the next section, we will provide some strategies that can help solve problem of parameter tuning.

2 Parameter tuning strategies

2.1 Review of common tuning strategies

Manual tuning: manual tuning is most straightforward but often time-consuming approach. Researches or practitioners manually select and adjust the DE algorithm's parameters based on their domain knowledge, intuition, or previous experience. While manual tuning can work well for well-understood problems, it may not be suitable for complex or large-scale tasks.

Grid search: Grid search involves systematically testing a predefined range of parameter values to identify the combination that results in the best performance. While this method can be effective, it can be computationally expensive, especially for problems with a large parameter space.

Random search: random search involves randomly selecting parameter values from specified ranges and conducting multiple optimizations runs. This approach can be more efficient than grid search and may uncover effective parameter settings. However, due to the random nature of this method, there is no guarantee that efficient parameter settings will be found [5].

Model-based optimization (MBO): MBO is a powerful approach that leverages statistical models to optimize complex and often expensive objective functions. In the context of tuning the parameters of algorithms, MBO provides a systematic and automated way to explore the parameter space and identify configurations that lead to optimal performance. However, MBO typically involves building a surrogate model, which can be computationally expensive, especially when dealing with high-dimensional parameter spaces. The model must be continuously updated and refined during optimization, adding to the computational load. In addition, selecting an appropriate surrogate model and configuring it can be a complex task, as surrogate models have their own hyperparameters that need to be tuned [6].

Adaptive algorithms: adaptive DE algorithms dynamically adjust their parameters during the optimization process. They use feedback from previous runs to fine-tune parameters, adapting to the specific characteristics of the problem. This approach can significantly reduce the need for manual tuning and improve the algorithm's ability to converge to optimal solution [7, 8].

Self-configuring algorithms: self-configuring algorithms go a step further, compared to adaptive algorithms. They not only adapt but also actively choose and configure different algorithm types or strategies during the optimization process. These algorithms can switch between entirely different optimization techniques, each with its set of parameters and strategies.

2.2 The idea of self-configuring algorithm

We developed a self-configuring differential evolution algorithm to solve parameter-tuning problem. Its core idea is to periodically reevaluate the choice of algorithm type and its associated parameters with selection favoring those configurations that have demonstrated superior performance [9].

The specific implementation is that for each predefined type of algorithm there is an associated probability of selecting that algorithm, the number of successful iterations of that type (S), and the number of unsuccessful iterations of that type (F). Successful iterations are those in which the algorithm has improved the best value found. Initially, all algorithm types have an equal probability of being chosen. Main loop is started when random algorithm type is selected based on each algorithm type's probability. During a predefined number of iterations (k), parameters S and F of the chosen algorithm type are updated. After that, the algorithm types are ranked according to the values of S and F (this article considers two types of ranking: based on difference between successful and unsuccessful iterations ($S - F$) and based solely on the number of successful iterations (S)). It is necessary to note that these types of rankings are quite similar, because with the same number of iterations they will give the same results. However, these ranking types are not identical as algorithm types are not selected the same amount of time. For instance, after completing the first k iterations, a particular algorithm is chosen once, while the remaining algorithms are not selected at all. Depending on the chosen type of ranking – the algorithm may be positioned either at the forefront or at the end of the ranking order.

Following the ranking, we apply an adjustment mechanism. A specific delta value is subtracted from each probability, while leaving a certain minimum probability. After that, the sum of the subtracted values is redistributed between the probabilities of the algorithm types according to their rank. Then we go back to the start of main loop, when new algorithm is selected based on recalculated probabilities. Self-configuring algorithm is shown in Algorithm 2 in form of pseudocode.

It is noteworthy that this self-configuration scheme is not limited to differential evolution alone. It could be applied to other optimization algorithms such as genetic algorithms (GA) or particle swarm optimization (PSO).

Algorithm 2: Self-configuring differential evolution

```
// Initialization
Set start array of DE types DE = [DE1, DE2 DE3] and associated parameters, F_DEi = 0, S_DEi = 0, P_DEi = 1 / length(DE);
Set number of iterations k;
Set delta;
Set minimum probability P_min;
Set maximum number of populations (M);
Form start population I_m in form of vectors [x^1, x^2, ..., x^NP];
Set number of population m = 0;
// Main loop
while m < M do
    Select DE_s from array DE based on probabilities P_DEi;
    for i to k do
        I_c = update current generation based on selected DE_s (see Algorithm 1);
        if best from I_c < I_m do
            S_DEs += 1;
        else
            F_DEs += 1;
```

```
        end
    end
    sum_sub = 0;
    for i in DE do
        if P_DEi − delta < P_min then
            sum_sub += P_DEi − P_min;
            P_DEi = P_min;
        else
            sum_sub += delta;
            P_DEi = P_DEi − delta;
        end
    end
    sort DE based on S_DEi − F_DEi or S_DEi;
    for i in length(DE) do
        P_DEi += (sum_sub * i) / sum(all indexes);
    end
    m += 1
end
Select best x from I_c
```

3 Numerical experiments

3.1 Conventional differential evolution

We chose three distinct variants of the differential evolution, which we have denoted as DE1, DE2, and DE3. Each variant was configured with specific parameters shown in the Table 1.

Table 1. Parameters of the chosen algorithms.

	F	**CR**	**Mutation strategy**
DE1	0.5	0.5	DE/rand/1
DE2	0.6	0.1	DE/rand/2
DE3	0.7	0.7	DE/current-to-rand/1

The test functions chosen were Rastrigin function (3), Schwefel function (4), Ackley function (5) and Rosenbrock (6) function, denoted as F1, F2, F3 and F4.

$$f(\boldsymbol{x}) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)]$$
$$x \in [-5.12, 5.12] \tag{3}$$

$$f(\boldsymbol{x}) = 418.9829d - \sum_{i=1}^{d} x_i sin\,(\sqrt{|x_i|}),$$
$$x \in [-500, 500] \tag{4}$$

$$f(\boldsymbol{x}) = -aexp(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - exp\left[\frac{1}{d}\sum_{i=1}^{d} cos\,(cx_i)\right] + a + ex\,p(1),$$
$$x \in [-32.768, 32.768] \quad (5)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$
$$x \in [-2.048, 2.048] \quad (6)$$

Each of these test functions was considered in two versions: one with two variables and another with ten variables. To ensure accuracy, we conducted 40 runs for each algorithm. The outcomes of these experiments are presented in Table 2, which displays reliabilities, and in Table 3, which provides the mean values of the best solution of each run over 40 repetitions.

Table 2. Reliabilities (%) of different differential evolution algorithms.

	DE1	**DE2**	**DE3**
F1, D = 2	100.0	95.0	100.0
F1, D = 10	100.0	100.0	0.0
F2, D = 2	100.0	72.5	100.0
F2, D = 10	100.0	100.0	0.0
F3, D = 2	100.0	72.5	100.0
F3, D = 10	100.0	100.0	100.0
F4, D = 2	100.0	100.0	100.0
F4, D = 10	0.0	0.0	100.0

Table 3. Mean values of the best solution of each run over 40 repetitions.

	DE1	**DE2**	**DE3**
F1, D = 2	0.00	0.00	0.00
F1, D = 10	0.00	0.00	14.24
F2, D = 2	0.00	0.02	0.00
F2, D = 10	0.00	0.00	687.58

F3, D = 2	0.00	0.01	0.00
F3, D = 10	0.00	0.00	0.00
F4, D = 2	0.00	0.00	0.00
F4, D = 10	2.84	2.36	0.00

3.2 Self-configuring differential evolution

We developed self-configuring algorithm described in section 2.8 and used DE1, DE2, DE3 as algorithm types from which our self-configuring algorithm can choose. The parameters of the algorithms were *delta = 0.1, minimal probability = 0.1* and *k = 10* for DE (S – F) and *delta = 0.05, minimal probability = 0.1* and *k = 10* for DE (S). In all algorithms the same maximum number of populations (M) were used, which allowed us to compare their effectiveness with each other. The results of these algorithms are shown in the Table 4 in form of reliabilities and in the Table 5 as the mean of the best solution of each run over 40 repetitions.

Table 4. Reliabilities (%) of self-configuring differential evolution algorithms compared to mean of plain differential evolution algorithms.

	Mean DE1, DE2, DE3	**DE (S – F)**	**DE (S)**
F1, D = 2	98.3	100.0	100.0
F1, D = 10	66.7	100.0	95.0
F2, D = 2	90.8	100.0	100.0
F2, D = 10	66.7	100.0	100.0
F3, D = 2	90.8	100.0	100.0
F3, D = 10	100.0	100.0	100.0
F4, D = 2	100.0	100.0	100.0
F4, D = 10	33.3	0.0	17.5

Table 5. Mean values of the best solution of each run over 40 repetitions of self-configuring differential evolution algorithms compared to mean of plain differential evolution algorithms.

	Mean DE1, DE2, DE3	**DE (S – F)**	**DE (S)**
F1, D = 2	0.00	0.00	0.00
F1, D = 10	4.75	0.00	0.01
F2, D = 2	0.01	0.00	0.00
F2, D = 10	229.19	0.00	0.00
F3, D = 2	0.00	0.00	0.00
F3, D = 10	0.00	0.00	0.00
F4, D = 2	0.00	0.00	0.00
F4, D = 10	1.73	0.38	1.84

3.3 Discussion of numerical results

As can be seen from the Tables 2 and 3, the performance of the differential evolution algorithm is notably influenced by its configuration parameters. Careful adjustments of these parameters can lead to remarkable results, enhancing the algorithm's ability to find optimal solutions. However, different problems may require distinct parameter settings to achieve optimal outcomes.

It is important to note that searching through all possible parameter combinations programmatically can be prohibitively expensive in terms of computational resources. Therefore, the challenge lies not only in identifying suitable parameter configurations but also in doing so efficiently, considering the specific characteristics of the problem at hand.

As observed in the Tables 4 and 5, the self-configuring algorithms consistently demonstrate performance on par with or superior to the reliability and mean values achieved by conventional algorithms. This observation implies that self-configuring algorithms hold a distinct advantage in optimization tasks. The data suggests that when employing a self-configuring algorithm, there is an increased likelihood of achieving superior results compared to use of conventional algorithms. This outcome underscores the advantages of leveraging self-configuring algorithms for optimization tasks.

4 Conclusions

In the course of our research, we explored the effectiveness of three different differential evolution algorithms, each configured with unique parameter sets, in the context of solving black-box optimization problems. Our experiments with conventional algorithms confirmed

that the performance of these algorithms exhibited a strong dependence on the chosen parameter configurations. It became evident that selecting the right combination of parameters was a non-trivial task, and misconfigurations could significantly hinder the algorithm's ability to find optimal solution.

In response to this parameter sensitivity, we devised a novel approach – a self-configuring algorithm. This adaptive algorithm periodically reassesses its choice of algorithm type and parameter values during the optimization process. What we observed was that, in the majority of cases, this self-configuring algorithm consistently matched or outperformed the traditional differential evolution variants. This performance boost demonstrated the potential of self-configuring algorithms in not only mitigating the parameter sensitivity issue but also significantly enhancing the overall effectiveness of black-box problem-solving techniques.

The principle of this self-configuration algorithm is not specific only to differential evolution as it can be used with other optimization algorithms.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

References

1. L.M. Rios, N.V. Sahinidis, Journal of Global Optimization **56**, 1247-1293 (2013)
2. R. Storn, K. Price, Journal of global optimization **11**, 341-359 (1997)
3. S. Das, P.N. Suganthan, IEEE Transactions on Evolutionary Computation **15**, 4-31 (2011)
4. A.W. Mohamed, A.A. Hadi, A.K. Mohamed, IEEE Access **9**, 68629-68662 (2021)
5. J. Bergstra, R. Bardenet, Y. Bengio, B. Kegl, *Algorithms for hyper-parameter optimization*, Advances in Neural Information Processing Systems, 2546-2554 (2011)
6. F. Hutter, H. H. Hoos, K. Leyton-Brown, *Sequential model-based optimization for general algorithm configuration*, Learning and Intelligent Optimization, 507-523 (2011)
7. M.G.H. Omran, A. Salman, A.P. Engelbrecht, *Self-adaptive differential evolution,* International conference on computational and information science, 192-199 (2005)
8. J. Brest, Soft Computing **11**, 617-629 (2007)
9. E. Sopov, Self-configuring Ensemble of Multimodal Genetic Algorithms. In: Merelo, J.J., et al. Computational Intelligence. IJCCI 2015. Studies in Computational Intelligence, vol 669. Springer, Cham (2017)