BLEND CROSSOVER MODIFICATION IN DIFFERENTIAL EVOLUTION: THE DE_D_BLEND APPROACH

A PREPRINT

Dimitar Nedanovsli
Department of Computer Science
Cranberry-Lemon University
Pittsburgh, PA 15213
dnedanovski@gmail.com

Svetoslav Nenov
Department of Mathematics
University of Chemical Technology and Metallurgy
Sofia, Bulgaria
nenov@uctm.edu

Dimitar Pilev
Department of Informatics
University of Chemical Technology and Metallurgy
Sofia, Bulgaria
pilev@uctm.edu

May, 2025

ABSTRACT

Differential Evolution (DE) is a widely used, population-based, stochastic metaheuristic optimization algorithm introduced by Storn and Price in 1995. It is especially effective for solving nonlinear, non-differentiable, and multimodal optimization problems in continuous domains.

In this short note, we propose a modification to the classical crossover technique in DE. We also present experimental results based on the CEC2017 test functions to demonstrate the performance of the proposed approach.

Keywords Differential Evolution, Crossover, (Diagonal) Blend Crossover

1 Introduction

Differential Evolution (DE) is a widely used, population-based stochastic metaheuristic optimization algorithm, introduced by Storn and Price in 1995. It is effective for solving nonlinear, non-differentiable, and multimodal optimization problems in continuous domains.

Formally, let $f : D \to \mathbb{R}$ be a given function, where $D \subseteq \mathbb{R}^d$ is the search domain (usually a cuboid). The goal is to find the global minimum x^* of f in D.

The DE method involves several key steps: initialization, mutation, crossover, and selection.

In this article, we present a simple yet effective modification to the classical crossover step, termed the blend crossover (or 'DE_d_blend'). We describe the modification, discuss its motivation, and provide results from numerical experiments.

All scripts and results are written using Python 3.12 and are publicly available on GitHub (see [citation]). Our main script is based on SciPy's '_differentialevolution.py' package, which is also available in [citation].

1.1 Crossover

In standard **binomial crossover** (also known as uniform crossover), the trial vector $\mathbf{u} = (u_1, u_2, \ldots, u_d)$ is constructed as follows:

$$u_j = \begin{cases} v_j, & \text{if rand}_j < CR \text{ or } j = j_{\text{rand}}, \\ x_j, & \text{otherwise,} \end{cases} \qquad j = 1, 2, \ldots, d, \tag{1}$$

where $\mathbf{v} = (v_1, v_2, \ldots, v_d)$ is the mutant vector, $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is the candidate (parent) vector, $CR \in [0, 1]$ is the (initially given) crossover probability, $j_{\text{rand}} \in \{1, 2, \ldots, d\}$ is a randomly chosen index to ensure at least one component of $\mathbf{x}$ is inherited from the mutant, i.e. $\mathbf{u} \neq \mathbf{x}$.

The classical crossover can be visualized in geometric terms:

- In **2D** (a rectangle), the candidate and mutant are two opposite vertices within the axis aligned rectangle. The trial vector's possible positions are either exactly all vertex, excluding candidate.
- In **3D** (a parallelepiped), the trial vector can occupy one of the $2^3 = 8$ corners of the axis aligned parallelepiped defined by the candidate and mutant vectors, as each coordinate independently chooses between the parent and mutant value.

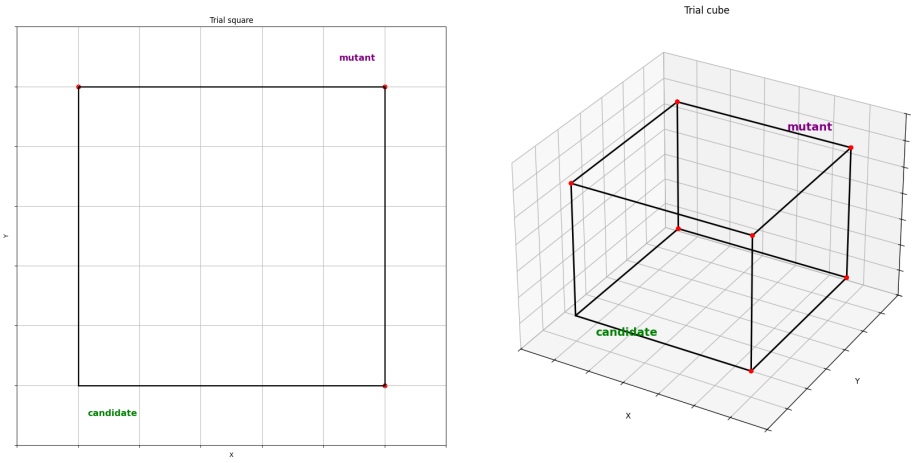

Figure 1: Classical crossover: Trial square and qube

The **blend crossover** operator generalizes the classical differential evolution (DE) crossover by allowing for weighted averaging (blending) between the candidate vector and the mutant vector over selected coordinates. First, we define the prime trial vector

$$\mathbf{u}' = p_{\text{blend}}\mathbf{x} + (1 - p_{\text{blend}})\mathbf{v},$$

where $\mathbf{x}$ and $\mathbf{v}$ are the candidate and mutant vectors, respectively, and $p_{\text{blend}} \in [d_1, d_2] \subseteq (0, 1]$ is a randomly chosen blending coefficient for the current population (i.e. population on current iteration). Here $[d_1, d_2]$ is initially given blend interval (i.e. we accept dither of p_{blend} in this interval).

If $p_{\text{blend}} = 1$, the trial is identical to the candidate (no update). For $0 < p_{\text{blend}} < 1$, the result is a point on the "main" diagonal of the cuboid defined by the candidate and mutant, i.e., a convex combination of the two.

The resulting point $\mathbf{u}' = (u'_1, u'_2, \ldots, u'_d)$ thus lies on the diagonal of a cuboid within the hypercube defined by the two parent vectors.

The next step is to either accept $\mathbf{u} = \mathbf{u}'$ as the trial vector with a given probability, or to project $\mathbf{u}'$ onto a lower-dimensional face (vertex, edge, or 2-face, ...) of the trial hypercube. The selection is governed by the crossover rate parameter CR:

- If $CR = 0$, then $\mathbf{u} = \mathbf{u}'$ (the blend is used directly).
- If $CR = 1$, $\mathbf{u}'$ is projected to a randomly chosen vertex, recovering the classical discrete crossover. Here we need exactly d orthogonal projections onto faces with one small dimension to move the prime trial on the diagonal of a d-dimensional cuboid to a vertex – the trial.

- For $0 < CR < 1$, the algorithm probabilistically projects $\mathbf{u}'$ onto a $(d-1)$-face, ... , 2-face, 1-face (edge), or vertex according to preset or user-defined probabilities.

The obtained projection is the trial $\mathbf{u}$.

In such a way the crossover constant becomes a parameter controlling the probability of projection depth, not just a per-coordinate mask. Indeed, if we consider projections on any k-face of cuboid, $k = 0, 1, \ldots, d-1$ with initially given probabilities $p_0, \ldots, p_{d-1}$ (such that $\sum p_k = 1$), then (continuing with the idea for convex combination) we calculate

$$P_k = CR\, p_k, \quad \text{prob. to project on } k\text{-face}, \; k = 0, 1, \ldots, d-1,$$

and one probability to stay on diagonal: $P_d = 1 - CR$. Again $\sum P_k = 1$.

In our experiments, for any dimension d, we will consider projections on k-face of cuboid, $k = 0, 1, 2$ and the probabilities for projecting to vertices, edges, and faces are fixed at 0.5, 0.3, and 0.2, respectively; CR dithers in $[0.6, 0.9]$. Then: $P_0 = 0.35$, $P_1 = 0.21$, $P_2 = 0.14$, and $P_3 = 0.3$ if $CR = 0.7$. We do not consider other cases in the article.

Combining all together, we added/modifing the following pseudocodes to differential evaluation algorithm: Algorithm 1 replaces classical DE mutation function and Algorithm sumarize all the above.

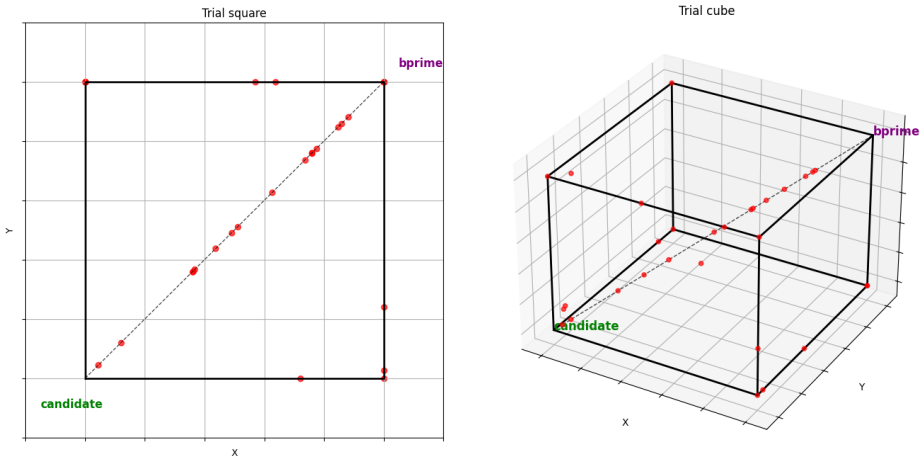

Figure 2: Blend crossover: Possible trials in square and qube

Algorithm 1 MUTATE(candidate)

```
rng ← random number generator
samples ← Select 5 samples (excluding candidate) from population
if strategy is 'currenttobest1exp' or 'currenttobest1bin' then
    bprime ← mutation_func(candidate, samples)
else
    bprime ← mutation_func(samples)
end if
trial ← DCROSSOVER(bprime, population[candidate], p_blend, crossover_probability, rng)
return trial
```

Algorithm 2 DCROSSOVER($\mathbf{x}, \mathbf{v}, p_{\text{blend}}$, crossover, base_probs, p_b, rng)

Require: Candidate vector $\mathbf{x}$, mutant vector $\mathbf{v}$, blend coefficient range $p_{\text{blend}} \in [d_1, d_2]$, crossover rate crossover $\in [0, 1]$, geometric probabilities base_probs $= [p_{\text{vertex}}, p_{\text{edge}}, p_{\text{face}}]$, random number generator rng

$d \leftarrow$ dimension of $\mathbf{x}$
$\mathbf{u}' \leftarrow p_b\mathbf{x} + (1 - p_b)\mathbf{v}$ ▷ blend on cuboid diagonal
if crossover ≤ 0 **then**
  return $\mathbf{u}'$ ▷ no projection, use diagonal blend
else if crossover ≥ 1 **then**
  $\mathbf{mask} \leftarrow rng$.choice$(\{0, 1\}, d)$
  return $\mathbf{mask} \odot \mathbf{x} + (1 - \mathbf{mask}) \odot \mathbf{v}$ ▷ random vertex: classic DE crossover
else
  probs $\leftarrow$ [crossover $\cdot\, p_{\text{vertex}}$, crossover $\cdot\, p_{\text{edge}}$, crossover $\cdot\, p_{\text{face}}$, $1 -$ crossover]
  probs $\leftarrow$ probs$/\sum$ probs
  onto_types $\leftarrow$ [vertex, edge, face, diagonal]
  onto $\leftarrow rng$.choice(onto_types, $p =$ probs)
  if onto $=$ diagonal **then**
    return $\mathbf{u}'$
  else if onto $=$ vertex **then**
    $\mathbf{mask} \leftarrow rng$.choice$(\{0, 1\}, d)$
    return $\mathbf{mask} \odot \mathbf{x} + (1 - \mathbf{mask}) \odot \mathbf{v}$
  else if onto $=$ edge **then**
    $j \leftarrow rng$.randint$(1, d)$
    for $i = 1$ **to** d **do**
      if $i = j$ **then**
        $u_i \leftarrow u'_i$
      else
        $u_i \leftarrow rng$.choice$([x_i, v_i])$
      end if
    end for
    return $\mathbf{u} = (u_1, \ldots, u_d)$
  else if onto $=$ face **then**
    $J \leftarrow$ randomly select 2 distinct coordinates from $\{1, \ldots, d\}$
    for $i = 1$ **to** d **do**
      if $i \in J$ **then**
        $u_i \leftarrow u'_i$
      else
        $u_i \leftarrow rng$.choice$([x_i, v_i])$
      end if
    end for
    return $\mathbf{u} = (u_1, \ldots, u_d)$
  end if
end if

Here all notations are understandable, except rng – All random choices in the DCROSSOVER operator (subspace selection, mask generation, coordinate selection) are governed by the `_differentialevolution.py` random number generator `rng`, ensuring both diversity and reproducibility in the crossover process.

2 Numerical Experiments

In Differential Evolution (DE), the *best-so-far* at iteration k refers to the best objective function value found up to and including iteration k in a single run. When multiple independent runs are executed, the median of the best-so-far values at each iteration is reported. This median curve provides a robust summary of the optimizer's typical progress, as it is less sensitive to outlier runs than the mean or the single best run.

Function	N	Wilcoxon p	T-test p	T-statistic	Conclusion
cec2017_f1	51	0.5486	0.346	-0.9513	No significant diff.
cec2017_f2	51	0.01299	0.01848	-2.435	DE significantly worse than DE_d_blended
cec2017_f3	51	0.216	0.1906	-1.327	No significant diff.
cec2017_f4	51	0.0469	0.02565	-2.3	DE significantly worse than DE_d_blended
cec2017_f5	51	0.2687	0.2344	-1.204	No significant diff.
cec2017_f6	51	0.5486	0.01868	-2.431	DE worse than DE_d_blended (partially)
cec2017_f7	51	0.00844	0.07928	1.791	DE **better** than DE_d_blended (partial sig.)
cec2017_f8	51	0.157	0.1321	1.531	No significant diff.
cec2017_f9	51	0.08458	0.08318	-1.768	No significant diff.
cec2017_f10	51	0.3989	0.2048	-1.285	No significant diff.
cec2017_f11	51	0.03037	0.005585	-2.896	DE significantly worse than DE_d_blended
cec2017_f12	51	6.152e-10	2.206e-15	-11.31	DE significantly worse than DE_d_blended
cec2017_f13	51	2.011e-06	0.007221	-2.801	DE significantly worse than DE_d_blended
cec2017_f14	51	4.416e-09	1.162e-08	-6.817	DE significantly worse than DE_d_blended
cec2017_f15	51	1.021e-08	7.899e-06	-4.982	DE significantly worse than DE_d_blended
cec2017_f16	51	0.109	0.5062	0.6695	No significant diff.
cec2017_f17	51	0.001002	0.1435	1.486	DE **better** than DE_d_blended (partial sig.)
cec2017_f18	51	6.152e-10	1.81e-10	-7.976	DE significantly worse than DE_d_blended
cec2017_f19	51	8.181e-09	4.606e-06	-5.138	DE significantly worse than DE_d_blended
cec2017_f20	51	0.0007652	0.01388	2.55	DE **significantly better** than DE_d_blended
cec2017_f21	51	0.1411	0.2626	-1.133	No significant diff.
cec2017_f22	51	0.8293	0.5027	-0.6752	No significant diff.
cec2017_f23	51	0.07191	0.02514	-2.308	DE worse than DE_d_blended (partially)
cec2017_f24	51	0.5178	0.6797	0.4153	No significant diff.
cec2017_f25	51	0.5931	0.8714	0.1627	No significant diff.
cec2017_f26	51	0.05706	0.1593	1.429	No significant diff.
cec2017_f27	51	4.57e-06	9.887e-07	-5.576	DE significantly worse than DE_d_blended
cec2017_f28	51	0.1542	0.01129	-2.631	DE worse than DE_d_blended (partially)
cec2017_f29	51	2.177e-07	3.386e-08	-6.52	DE significantly worse than DE_d_blended
cec2017_f30	51	1.254e-05	0.03092	-2.221	DE significantly worse than DE_d_blended

Table 1: Wilcoxon and paired t-test results for DE and DE_d_blended on CEC2017 functions (number of separate runs= 51, $d = 10$, using $\alpha = 0.05$ for significance).

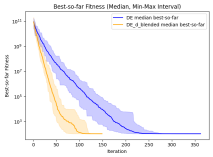

Figure 3: f_1

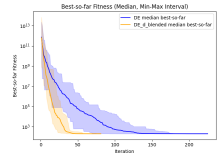

Figure 4: f_2

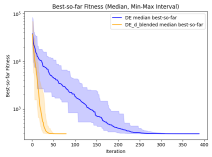

Figure 5: f_3

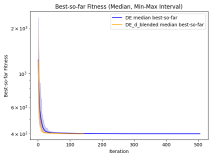

Figure 6: f_4

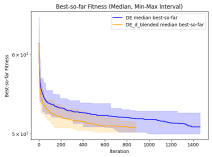

Figure 7: f_5

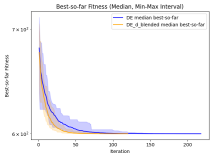

Figure 8: f_6

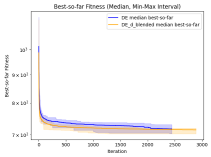

Figure 9: f_7

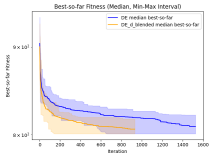

Figure 10: f_8

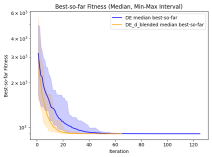

Figure 11: f_9

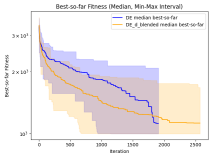

Figure 12: f_{10}

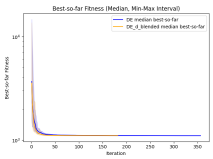

Figure 13: f_{11}

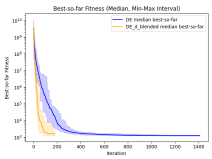

Figure 14: f_{12}

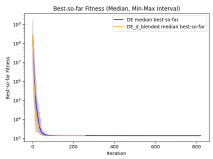

Figure 15: f_{13}

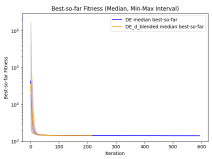

Figure 16: f_{14}

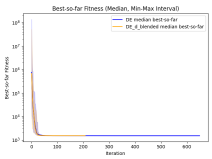

Figure 17: f_{15}

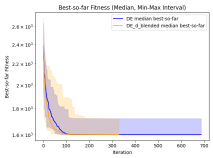

Figure 18: f_{16}

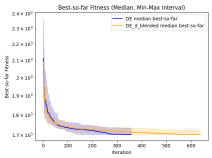

Figure 19: f_{17}

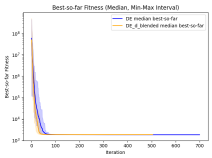

Figure 20: f_{18}

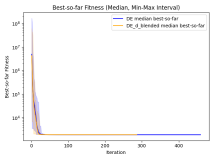

Figure 21: f_{19}

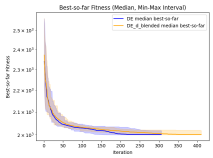

Figure 22: f_{20}

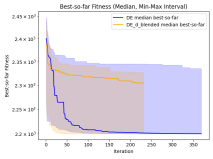

Figure 23: f_{21}

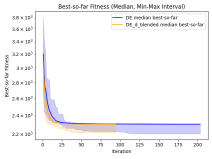

Figure 24: f_{22}

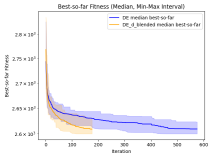

Figure 25: f_{23}

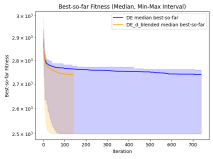

Figure 26: f_{24}

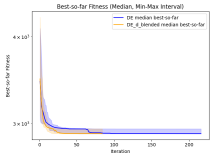

Figure 27: f_{25}

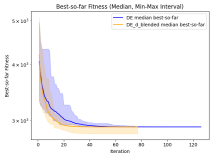

Figure 28: f_{26}

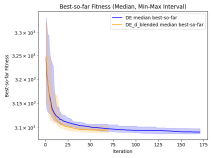

Figure 29: f_{27}

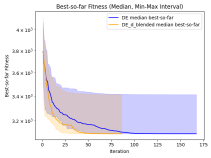

Figure 30: f_{28}

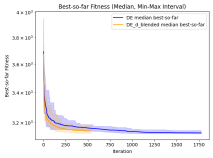

Figure 31: f_{29}

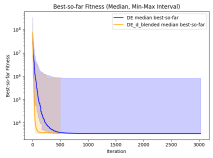

Figure 32: f_{30}

3 Comparison with other methods: L-SHADE

In this section, we compare two methods: the classical L-SHADE (see [1, 2]) and its variant with blended crossover (L-SHADE-BC).

Our evaluation follows the guidelines of the CEC 2017 benchmark competition, see [3, 4]. For all problems, the search space is $[-100, 100]^d$. If the difference between the best solution found and the optimal solution is less than or equal to 10^{-11}, the error (score) is reported as 0.00; exact matches are denoted as integer. Each problem was tested in dimensions $d = 10$ and $d = 30$. For each problem and dimension, 51 independent runs were performed, and average performance was evaluated. All results (10-dimensional case, for the first 10 CEC functions) are sumarized in Table 2. The full statistics and pickle archive of all runs are available in HitHub repository [], `subdirectory cec2017_f1_f10_d10`.

For functions f_1, f_2, f_3, f_6, and f_9, our method achieves both a lower mean error (*Mean Fun*) and standard deviation (*St Dev Fun*), while also requiring significantly fewer iterations (*nit*) compared to the baseline. For functions f_5 and f_8, altough our approach requires much more iterations, it consistently reaches the true minimum in all 51 runs. In the case of f_7, the standard deviation of the results is approximately zero (maybe indicating extremely stable convergence behavior across all 51 runs).

Statistical significance was assessed using the Wilcoxon rank-sum test (Mann–Whitney U test) with a significance threshold of $p < 0.05$. For all functions, the difference between the algorithms was found to be statistically significant, see Table 3. Each table entry shows the mean and standard deviation of the error (the difference between the best fitness found in each run and the optimal value). The best result for each problem is shown in bold. The symbols $+$, $-$, and $\equiv$ indicate whether the given algorithm performed significantly better ($+$), significantly worse ($-$), or not significantly different ($\equiv$) compared to L-SHADE according to the Wilcoxon rank-sum test ($p < 0.05$).

Table 2: Selected statistics for CEC2017 benchmark functions f_1–f_{10}, $d = 10$.

Function	Algorithm	Mean Fun	Median Fun	St Dev Fun	Median Nit	Median NFEV
f_1	L-SHADE	100.0000000026	100.0000000026	7.25e-10	417	410612
f_1	L-SHADE-BC	100.0000000017	100.0000000017	4.54e-10	379	373873
f_2	L-SHADE	200.0000000004	200.0000000004	1.68e-10	331	327298
f_2	L-SHADE-BC	200.0000000001	200.0000000002	5.69e-11	277	274678
f_3	L-SHADE	300.0000000027	300.0000000029	7.07e-10	367	362247
f_3	L-SHADE-BC	300.0000000019	300.0000000019	5.36e-10	272	269793
f_4	L-SHADE	400.0000000025	400.0000000025	6.31e-10	447	439534
f_4	L-SHADE-BC	400.0000000014	400.0000000013	3.20e-10	554	542105
f_5	L-SHADE	500.0000000035	500.0000000035	1.05e-09	5902	4659540
f_5	L-SHADE-BC	500	500	0	9906	6739134
f_6	L-SHADE	600.0000000109	600.0000000113	2.15e-09	2029	1866602
f_6	L-SHADE-BC	600.0000000092	600.0000000095	1.55e-09	585	571650
f_7	L-SHADE	710.1636478201	710.3669207765	1.44e+00	8920	6289550
f_7	L-SHADE-BC	710.3669207724	710.3669207724	2.27e-13	19503	9628561
f_8	L-SHADE	800.0000000034	800.0000000035	7.38e-10	5730	4553705
f_8	L-SHADE-BC	800	800	0	10100	6823321
f_9	L-SHADE	900.0000000031	900.0000000030	7.42e-10	378	372905
f_9	L-SHADE-BC	900.0000000022	900.0000000021	6.55e-10	217	215934
f_{10}	L-SHADE					
f_{10}	L-SHADE-BC					

Table 3: Comparison of L-SHADE and L-SHADE-BC on CEC2017 functions ($d = 10$) with Wilcoxon rank-sum test statistics.

Function	Mean		Median		U statistic	p-value	Conclusion
	L-SHADE	L-SHADE-BC	L-SHADE	L-SHADE-BC			
f_1	100.000000	100.000000	100.000000	100.000000	2210.0	1.17×10^{-9}	+
f_2	200.000000	200.000000	200.000000	200.000000	2503.0	8.65×10^{-16}	+
f_3	300.000000	300.000000	300.000000	300.000000	2118.0	4.55×10^{-8}	+
f_4	400.000000	400.000000	400.000000	400.000000	2466.0	6.34×10^{-15}	+
f_5	500.000000	500.000000	500.000000	500.000000	2601.0	1.39×10^{-20}	+
f_6	600.000000	600.000000	600.000000	600.000000	1970.0	7.56×10^{-6}	+
f_7	710.163648	710.366921	710.366921	710.366921	2550.0	4.03×10^{-19}	−
f_8	800.000000	800.000000	800.000000	800.000000	2601.0	1.39×10^{-20}	+
f_9	900.000000	900.000000	900.000000	900.000000	2059.0	3.91×10^{-7}	+
f_{10}							

4 Numerically anstable domains in initial conditions space

References

[1] Ryoji Tanabe and Alex S. Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation (CEC)*, pages 71–78. IEEE, 2013. doi:10.1109/CEC.2013.6557555.

[2] Ryoji Tanabe and Alex S. Fukunaga. Improving the search performance of shade using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1658–1665. IEEE, 2014. doi:10.1109/CEC.2014.6900387.

[3] N. H. Awad, M. Z. Ali, P. N. Suganthan, J. J. Liang, and B. Y. Qu. Problem definitions and evaluation criteria for the cec 2017 special session and competition on single objective real-parameter numerical optimization. Technical report, Nanyang Technological University, Singapore and Jordan University of Science and Technology, Jordan and Zhengzhou University, China, 2016. Modified on October 15th 2016.

[4] Cec 2017 python. `https://github.com/tilleyd/cec2017-py`, 2022. commit hash or version, if relevant.

[5] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. Technical report, Technical University of Wroclaw, Wroclaw, Poland, 2005. URL `http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf`. 3 kwietnia 2005.

[6] Ji Qiang and Chad Mitchell. A unified differential evolution algorithm for global optimization. *Journal of Global Optimization*, 2016.

[7] Jingqiao Zhang and Arthur C. Sanderson. Jade: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009. doi:10.1109/TEVC.2009.2014613.

[8] Eugene Semenkin Vladimir Stanovov. Success rate-based adaptive differential evolution l-srtde for cec 2024 competition. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2024. doi:10.1109/CEC60901.2024.10611907.

[9] Bilal, Millie Pant, Hira Zaheer, Laura Garcia-Hernandez, and Ajith Abraham. Differential evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence*, 90:103479, 2020. doi:10.1016/j.engappai.2020.103479.

[10] Ran Cheng et al. Advancements in multimodal differential evolution: A comprehensive review. *arXiv preprint arXiv:2504.00717*, 2024. URL `https://arxiv.org/abs/2504.00717`.

[11] Shivani Dikshit Chauhan1. Multimodal differential evolution: A detailed review. *SSRN Electronic Journal*, 2024. doi:10.2139/ssrn.5044007. URL `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5044007`.

[12] A. Layeb. Differential evolution algorithms with novel mutations, adaptive parameters, and weibull flight operator. *Soft Computing*, 28:7039–7091, 2024. doi:10.1007/s00500-023-09561-3.