Good Parameters for Differential Evolution

By
Magnus Erik Hvass Pedersen
Hvass Laboratories

Technical Report no. HL1002
2010

Abstract

The general purpose optimization method known as Differential Evolution (DE) has a number of parameters that determine its behaviour and efficacy in optimizing a given problem. This paper gives a list of good choices of parameters for various optimization scenarios which should help the practitioner achieve better results with little effort.

Keywords: Numerical optimization, differential evolution, parameters.

1 Introduction

The optimization method known as Differential Evolution (DE) was originally introduced by Storn and Price [1] and offers a way of optimizing a problem without using its gradient. This is particularly useful if the gradient is difficult or even impossible to derive.

DE maintains a population of agents which are iteratively combined and updated using simple formulae to form new agents. The practitioner has to set a number of behavioural parameters that influence the performance of this process, see for example Storn et al. [2] [3], Liu and Lampinen [4], and Zaharie [5]. There has been a trend in recent years to try and make the DE parameters automatically adapt to new problems during optimization, hence alleviating the need for the practitioner to select the parameters by hand, see for example Price et al. [3], Liu and Lampinen [6], Qin et al. [7] [8], and Brest et al. [9]. But these DE variants with so-called adaptive parameters just introduce new parameters that must then be set by the practitioner and has therefore merely deferred this difficult issue without actually eliminating it. Furthermore, we have previously demonstrated that basic DE variants with properly tuned parameters have comparable performance [10] [11].

This paper gives the practitioner a table of DE parameters that have been tuned for different optimization scenarios.

2 Differential Evolution

Consider a fitness (or cost, error, objective) function:

$$f : \mathbb{R}^n \to \mathbb{R}$$

To minimize the fitness function f find $\vec{a} \in \mathbb{R}^n$ so that:

$$\forall \vec{b} \in \mathbb{R}^n : \; f(\vec{a}) \leq f(\vec{b})$$

Then $\vec{a}$ is called a global minimum for the function f. It is usually not possible to pinpoint the global minimum exactly in optimization and candidate solutions with sufficiently good fitness are deemed acceptable for practical reasons.

In DE the candidate solutions are called agents and are denoted $\vec{x} \in \mathbb{R}^n$. They are initially placed at random positions in the search-space and are iteratively updated by combining a number of agents from the population and keeping the new agent if it improves on the fitness.

Small changes to the DE implementation can cause dramatic changes in the behavioural parameters that cause good optimization performance. The parameters given in this paper have been tuned for the DE/rand/1/bin algorithm in figure 1. If your DE implementation differs from this you may need to alter it to use the parameters listed here.

3 Meta-Optimization

The DE parameters in table 1 have been found by way of meta-optimization, that is, the use of another overlying optimizer to tune the DE parameters for different optimization scenarios. The concept is depicted in figure 2 and described in detail in [10].

The DE parameters have been tuned for the benchmark problems in table 2 using various dimensionalities and optimization run-lengths. Note that the optimum has been displaced according to the values in table 3 to avoid unintended attraction of the DE agents to zero which also happens to be the global optimum of most of these benchmark problems. All 12 benchmark problems have been used in meta-optimization to yield behavioural parameters that should work well in general, although for some of the larger meta-optimization scenarios, e.g. the 100 dimensional cases, only the Ackley, Rastrigin, Rosenbrock and Schwefel1-2 problems were used so as to save computation time.

Time usage for meta-optimization of the smallest problem configurations (2 dimensions and 400 fitness evaluations) were mere seconds while up to 24 hours were used for the larger problem configurations when executed on a 1.6 GHz Intel CPU. Using a modern multi-core CPU would decrease the time usage considerably and is readily supported in the source-code linked to below.

4 Example Usage

If you need to optimize a problem using few fitness evaluations, say, a 4-dimensional problems using 100 fitness evaluations, or a 1,000-dimensional problem using 30,000 fitness evaluations, then DE may not be the right choice of optimizer. Instead you may want to use optimizers that were specifically designed for short optimization runs, see e.g. Pattern Search (PS) and Local Unimodal Sampling (LUS) in [10].

Now assume you are tasked with optimizing a series of problems in 40 dimensions each and you can perform 500,000 fitness evaluations on each problem, what DE parameters should you use? Consulting table 1 we see that this exact scenario is not listed. The practitioner will then try with the closest match and if that does not yield satisfactory results then try the next closest match, etc. In this case the closest match seems to be the parameters tuned for 30 dimensions and 600,000 fitness evaluations:

$$NP = 75,\ CR = 0.8803,\ F = 0.4717$$

where NP is the number of agents in the population. Using these parameters in optimizing the benchmark problems results in table 4 and figures 3 and 4. On most of these problems the results are close to the optimal fitness values of zero, but the Rastrigin, Rosenbrock and Schwefel2-21 problems were not quite solved. Then the practitioner would have to decide if other parameters from table 1 should be tried, perhaps the parameters tuned for 50 dimensions and 100,000 fitness evaluations, or 20 dimensions and 400,000 fitness evaluations. If those fail as well then the practitioner would need to either meta-optimize the DE parameters specifically for these remaining problems or use another optimization method. (All these problems are optimized well in [12].)

5 Conclusion

This paper presented a table of DE parameters that may be used by the practitioner as a first choice when optimizing new problems. The parameters were tuned (meta-optimized) to perform well on several benchmark problems with various dimensionalities and optimization run-lengths.

6 Source-Code

Source-code implemented in the C# programming language and used in the experiments in this paper can be found in the SwarmOps library on the internet address: `http://www.Hvass-Labs.org/`

References

[1] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.

[2] R. Storn. On the usage of differential evolution for function optimization. In *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523, Berkeley, CA, USA, 1996.

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

[4] J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*, pages 11–18, Brno, Czech Republic, 2002.

[5] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, pages 62–67, Bruno, 2002.

[6] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

[7] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE congress on evolutionary computation (CEC)*, pages 1785–1791, 2005.

[8] 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*, 13:398–417, 2009.

[9] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[10] M.E.H. Pedersen. *Tuning & Simplifying Heuristical Optimization.* PhD thesis, School of Engineering Sciences, University of Southampton, England, 2010.

[11] M.E.H. Pedersen and A.J. Chipperfield. Tuning differential evolution for artificial neural networks. In S.J. Kwon, editor, *Artificial Neural Networks.* Nova Publishers, 2011.

[12] M.E.H. Pedersen. Good parameters for particle swarm optimization. Technical Report HL1001, Hvass Laboratories, 2010.

```
• Initialize all agents $\vec{x} \in \mathbb{R}^n$ with random positions in the search-space:
      $\vec{x} \sim U(\vec{b}_{lo}, \vec{b}_{up})$
  where $\vec{b}_{lo}$ and $\vec{b}_{up}$ are the lower and upper boundaries of the search-space.
• Until a termination criterion is met, repeat the following:
    – For each agent $\vec{x}$ in the population do the following:
        ∗ Pick three agents $\vec{a}$, $\vec{b}$ and $\vec{c}$ from the population at random, they
          must be distinct from each other as well as from agent $\vec{x}$.
        ∗ Pick a random index $R \in \{1, \cdots, n\}$, where the highest possible
          value $n$, is the dimensionality of the problem to be optimized.
        ∗ Compute the agent's potentially new position $\vec{y} = [y_1, \cdots, y_n]$,
          by iterating over each $i \in \{1, \cdots, n\}$ as follows:
            · Pick $r_i \sim U(0, 1)$ for use in a stochastic choice next.
            · Compute the $i$'th element of the potentially new position $\vec{y}$:
                  $y_i = \begin{cases} a_i + F(b_i - c_i) & \text{, if } (i = R) \text{ or } (r_i < CR) \\ x_i & \text{, else} \end{cases}$
              Where the user-defined behavioural parameters are the dif-
              ferential weight $F$ and the crossover probability $CR$.
        ∗ Bound the position $\vec{y}$, that is, for all dimensions $i$ update $y_i$:
              $y_i \leftarrow \begin{cases} b_{lo_i} & , y_i < b_{lo_i} \\ b_{up_i} & , y_i > b_{up_i} \\ y_i & , else \end{cases}$
        ∗ If $(f(\vec{y}) < f(\vec{x}))$ then update the agent's position:
              $\vec{x} \leftarrow \vec{y}$
• Now the agent $\vec{x}$ from the population having the lowest fitness $f(\vec{x})$ is the
  best found position.
```

Figure 1: DE pseudo-code.

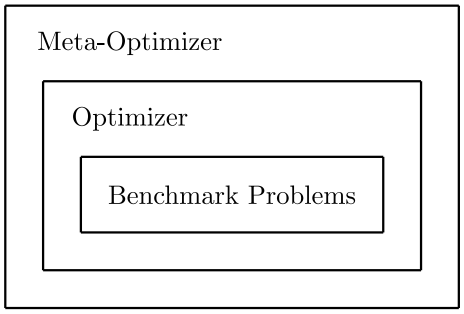

Figure 2: The concept of meta-optimization. Another optimization method is used as an overlying meta-optimizer for finding good behavioural parameters of DE, which in turn is used to optimize benchmark problems.

Problem Dimensions	Fitness Evaluations	DE Parameters		
		NP	CR	F
2	400	13	0.7450	0.9096
		10	0.4862	1.1922
2	4,000	24	0.2515	0.8905
		20	0.7455	0.9362
5	1,000	17	0.7122	0.6301
5	10,000	20	0.6938	0.9314
10	2,000	28	0.9426	0.6607
		12	0.2368	0.6702
10	20,000	18	0.5026	0.6714
20	40,000	37	0.9455	0.6497
20	400,000	35	0.4147	0.5983
30	600,000	75	0.8803	0.4717
50	100,000	48	0.9784	0.6876
100	200,000	46	0.9565	0.5824

Table 1: DE parameters for various problem configurations. The practitioner should select the DE parameters where the dimensionality and allowed number of fitness evaluations most closely match those of the optimization problem at hand. For some problem configurations multiple parameters are listed as they had almost the same optimization performance.

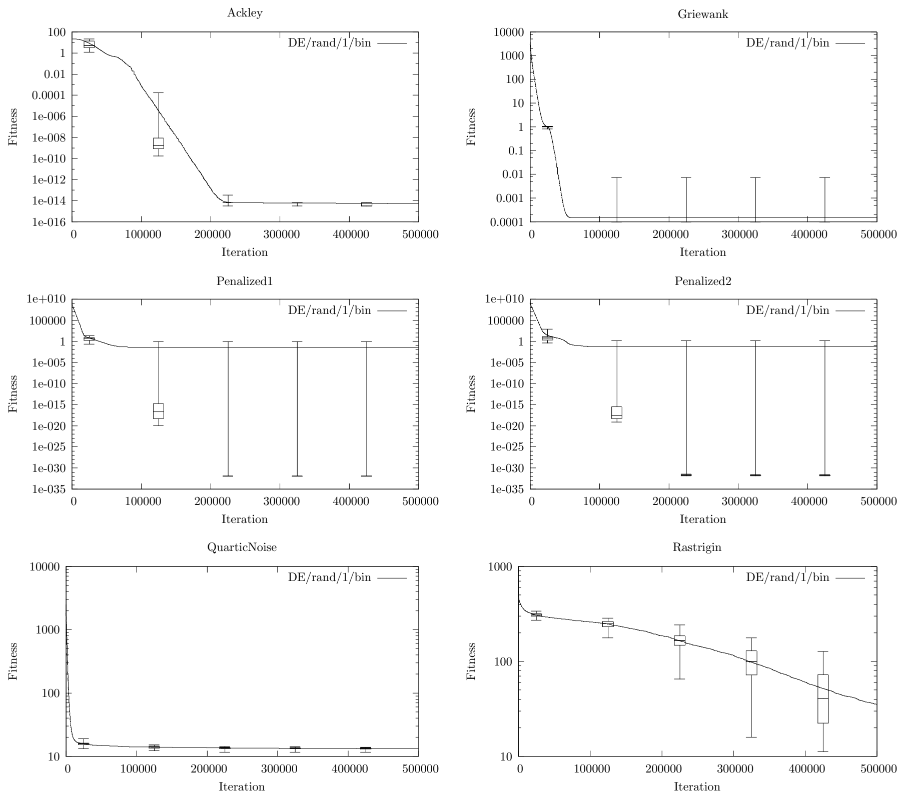

Figure 3: DE optimization performance. Plots show the mean fitness achieved over 50 optimization runs as well as the quartiles at intervals during optimization.

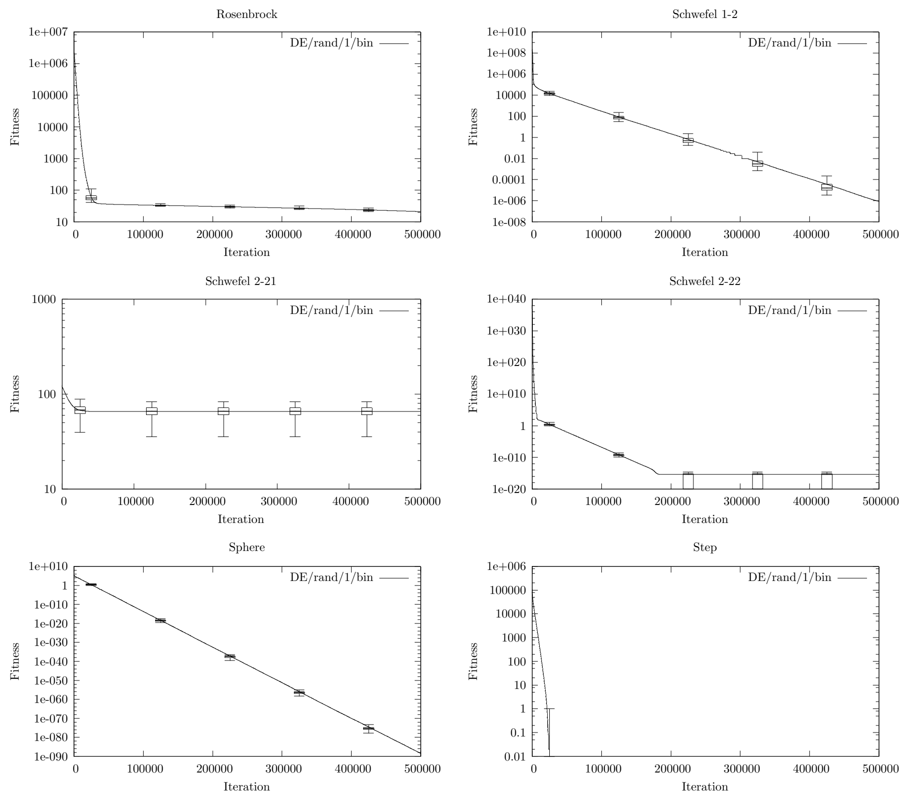

Figure 4: DE optimization performance. Plots show the mean fitness achieved over 50 optimization runs as well as the quartiles at intervals during optimization.

Ackley	$f(\vec{x}) = e + 20 - 20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$
Griewank	$f(\vec{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$
Penalized1	$f(\vec{x}) = \frac{\pi}{n}\Big(10 \cdot \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2 \cdot \left(1 + 10 \cdot \sin^2(\pi y_{i+1})\right) + (y_n - 1)^2\Big) + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$; $y_i = 1 + (x_i + 1)/4$; $u(x_i, a, k, m) = \begin{cases} k(-x_i - a)^m & , x_i < -a \\ 0 & , -a \leq x_i \leq a \\ k(x_i - a)^m & , x_i > a \end{cases}$
Penalized2	$f(\vec{x}) = 0.1\Big(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \cdot \left(1 + \sin^2(3\pi x_{i+1})\right) + (x_n - 1)^2 \cdot \left(1 + \sin^2(2\pi x_n)\right)\Big) + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$, with $u(\cdot)$ from above.
QuarticNoise	$f(\vec{x}) = \sum_{i=1}^{n}(i \cdot x_i^4 + r_i),\ r_i \sim U(0,1)$
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{n}\left(x_i^2 + 10 - 10 \cdot \cos(2\pi x_i)\right)$
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n-1}\left(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$
Schwefel1-2	$f(\vec{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$
Schwefel2-21	$f(\vec{x}) = \max\{\lvert x_i \rvert : i \in \{1, \cdots, n\}\}$
Schwefel2-22	$f(\vec{x}) = \sum_{i=1}^{n}\lvert x_i \rvert + \prod_{i=1}^{n}\lvert x_i \rvert$
Sphere	$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Step	$f(\vec{x}) = \sum_{i=1}^{n}\left(\lfloor x_i + 0.5 \rfloor\right)^2$

Table 2: Benchmark problems.

Problem	Initialization	Search-Space	Displacement δ
Ackley	$[15, 30]$	$[-30, 30]$	-7.5
Griewank	$[300, 600]$	$[-600, 600]$	-150
Penalized1	$[5, 50]$	$[-50, 50]$	0
Penalized2	$[5, 50]$	$[-50, 50]$	0
QuarticNoise	$[0.64, 1.28]$	$[-1.28, 1.28]$	-0.32
Rastrigin	$[2.56, 5.12]$	$[-5.12, 5.12]$	1.28
Rosenbrock	$[15, 30]$	$[-100, 100]$	25
Schwefel1-2	$[50, 100]$	$[-100, 100]$	-25
Schwefel2-21	$[50, 100]$	$[-100, 100]$	-25
Schwefel2-22	$[5, 10]$	$[-10, 10]$	-2.5
Sphere	$[50, 100]$	$[-100, 100]$	25
Step	$[50, 100]$	$[-100, 100]$	25

Table 3: Initialization ranges, search-space boundaries, and displacement values δ for the benchmark problems. Displacement is done by using an auxiliary fitness function $h(\vec{x}) = f(\vec{x} - \delta)$ to avoid unintended attraction of DE agents to the zero-position which happens to be the optimal solution for most of these problems.

Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
Ackley	5.45e-15	1.68e-15	3.11e-15	3.11e-15	6.66e-15	6.66e-15	6.66e-15
Griewank	1.48e-4	1.04e-3	0	0	0	0	7.4e-3
Penalized1	0.04	0.15	1.18e-32	1.18e-32	1.18e-32	1.18e-32	0.95
Penalized2	0.06	0.31	1.35e-32	1.72e-32	2.21e-32	2.71e-32	1.61
QuarticNoise	13.16	0.51	11.66	12.84	13.21	13.54	13.95
Rastrigin	35.21	25.5	8.95	17.91	24.38	45.18	104.44
Rosenbrock	21.1	1.91	15.89	19.86	20.88	21.98	25.41
Schwefel1-2	8.59e-7	1.2e-6	8.95e-8	2.28e-7	3.39e-7	9.25e-7	6.6e-6
Schwefel2-21	65.56	8.72	35.6	60.85	66.08	71.99	83.25
Schwefel2-22	4.09e-16	5.01e-16	0	0	4.44e-16	8.88e-16	2.22e-15
Sphere	3.87e-89	1.21e-88	1.78e-92	1.16e-90	3.21e-90	1.08e-89	6.89e-88
Step	0	0	0	0	0	0	0

Table 4: Optimization end results for DE when the benchmark problems have 40 dimensions and 500,000 fitness evaluations have been performed.