# A Comparison of Differential Evolution (DE/rand/1) with Stochastic Gradient Descent and Random Search

Judy Warner-Willich
CEMPS
University of Exeter
Exeter, UK
jtw210@exeter.ac.uk

## ABSTRACT

In this paper we implement differential evolution (DE), specifically DE/rand/1/bin, in Python and apply it to various optimization problems. We then compare the its performance against a random search algorithm and a stochastic gradient descent (SGD) algorithm. DE easily outperforms the other algorithms and shows good results on all test problems, even for a small number of iterations.

## KEYWORDS

Machine learning, differential evolution, optimization, evolutionary algorithms, direct search, minimisation, gradient descent

## 1 INTRODUCTION

Many real world sectors rely on complex decision making; from fine tuning machinery and automobile design, to civil engineering and telecommunications, optimisation of problem parameters is key to creating a successful design. Most real-life problems can be formulated in terms of an optimisation model, where multiple objectives and criteria must be met. Generally speaking, an optimisation problem is defined as follows:

Minimise $f(\mathbb{x})$

Subject to $\mathbb{x} \in \Omega$

Where $f : \Omega \to \mathbb{R}$ is the objective function that must be minimised, $\Omega$ is the decision space, and $\mathbb{x}$ is a candidate solution. Additional requirements and boundaries may be set as necessary.

Finding the best solution to a problem can prove incredibly difficult when performed manually by a human, as many parameters must be simultaneously accounted for. To solve this issue, optimisation algorithms can increase the speed and potential for solving these problems accurately. Multiple classes of optimisation algorithm exist including local search algorithms, like simulated annealing and tabu search, and population search algorithms, such as particle swam optimisation, and evolutionary algorithms (EAs). EAs use an iterative process to ‘evolve’ their solutions to improve the result. A potential solution(s) is initialised and the algorithm attempts to improve on each solution by varying it in a number of possible ways, including crossovers and mutations. If the new solution is indeed improved upon, determined via a fitness function, then it replaces the old solution. The repetition of the process can converge solutions towards a global optimum. Algorithms which evolve single solutions can run the risk of getting trapped in a local optimum and will never move towards the true, global optimum; a solution to this problem is to evolve many solutions side by side, or by occasionally allowing a ‘bad’ move, in the hope of getting closer to the true optimum.

Differential evolution (DE) is a parallel direct search method that evolves a population of solutions to solve numerical optimisation problems. It can find solutions for problems in any number of dimensions, and even on non-differentiable functions. The individuals in the population are mixed by adding the difference between two solutions to another, and evaluating the performance.

This paper will cover the implementation of the DE algorithm in Python, formulation of optimisation problems, and experimental results comparing the algorithm to a random search and a stochastic gradient descent (SGD) algorithm.

## 2 LITERATURE REVIEW

Some of the first evolutionary techniques in computer programming were proposed in the 1960s such as Fogel’s introduction of evolutionary programming [4,5], Schwefel and Rechenberg’s evolution strategies [6,7,8]. Over the following decades, other nature inspired algorithms were developed, including ant colony optimisation, particle swarm optimisation, and other forms of genetic algorithms.

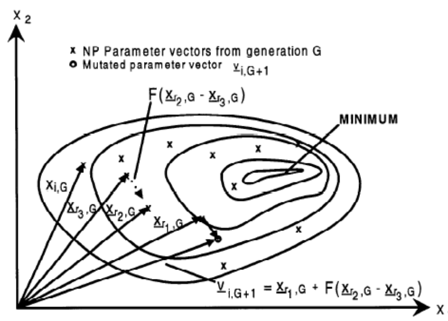
$x_2$
x NP Parameter vectors from generation G
o Mutated parameter vector $v_{i,G+1}$
$F(x_{r_2,G} - x_{r_3,G})$
MINIMUM
$x_{i,G}$
$x_{r_3,G}$
$x_{r_2,G}$
$x_{r_1,G}$
$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} - x_{r_3,G})$
$x_1$

**Figure I. An example of vector mutation in the DE algorithm, reproduced from Storn & Price (1997) [1].**

The differential evolution algorithm was proposed in 1996 by Storn and Price as a technical report [1,3], proposing a new solution for finding the global optimum of non-linear, non-differentiable and multi-modal functions. It was designed also with ease of use considered, so as to minimise the number of hyperparameters that a user would need to optimise for and make them easy to choose. The success of the new algorithm was shown at the First International Contest on Evolutionary Optimisation (ICEO) in 1996 where DE finished in third place [9]. Since that time, DE has emerged to become a popular algorithm for use in solving optimisation problems. In addition to the original algorithm, many variants have been proposed that continue to prove the success of the algorithm. An example of a vector mutation is shown in Figure 1.

The original algorithm takes a random vector in the population and mutates it via an addition of a difference vector, multiplied by a user-specified factor. Variants include using multiple difference vectors, and using the current best solution instead of a random one. These were all proposed in the original paper, however further advancements over the decades have provided new ways to use the algorithm. For example, in 2010 Chiang et. al [15] proposed a 2-Opt based DE algorithm which was confirmed to outperform the original DE algorithm. Other variants are surveyed and described in [16].

## 3 PROBLEM FORMULATION

Five test problems were used here: the Ackley function [10], a function proposed by David Ackley in 1987; the Rastrigin function [11] proposed in 1974 by Rastrigin and generalised to n-dimensions by Rudolph [12]; the Rosenbrock function [13], introduced in 1960 by Howard H. Rosenbrock; the Griewank function [14], introduced in 1981 by A. O. Griewank; and finally the Easom function. All of these test problems were designed for the purpose of testing optimisation algorithms.

**TABLE I.**
**Test problems used in the Experiments.**

| **Problem** | **Objective Function** | **Dimensions** | **Global minimum** | **Glbl. Opt. Location** |
|---|---|---|---|---|
| **F1** | Easom | 2 | -1 | (π,π) |
| **F2** | Ackley | 15 | 0 | (0,0,…,0) |
| **F3** | Ackley | 30 | 0 | (0,0,…,0) |
| **F4** | Rosenbrock | 15 | 0 | (1,1,…,1) |
| **F5** | Rosenbrock | 30 | 0 | (1,1,…,1) |
| **F6** | Rastrigin | 15 | 0 | (0,0,…,0) |
| **F7** | Rastrigin | 30 | 0 | (0,0,…,0) |
| **F8** | Griewank | 15 | 0 | (0,0,…,0) |
| **F9** | Griewank | 30 | 0 | (0,0,…,0) |

These four test problems provide different levels of solving difficulty to test the algorithm on. The Rosenbrock function is unimodal, with the global minima lying in a narrow valley; finding the valley is simple but it can be difficult to find the global minimum. The Easom function is also unimodal, and has its global minimum in a very small area relative to the search space. It is only a two-dimensional problem. The other three problems are multimodal in nature. The Ackley problem consists of a fairly flat outer region, with the minimum residing in a large dip at the centre. The outer region contains many local minima, which pose a risk for hillclimbing algorithms. The Griewank function contains many regularly distributed local minima, with a high complexity. Finally, the Rastrigin problem is similar to Griewank, with a high complexity multimodal landscape.

Details of the exact test problems used in the experiments are shown in Table 1. A range of dimensions for the problems (except Easom) were used to compare performance.

## 4 THE SEARCH ALGORITHM

```
x <- initialise_population(size=NP, dimensions=d)
u <- zeros array (size=NP, dimensions=d)
while (current_gen < max_gen) # stop after
                      max_gen generations
{
  for (i=1 -> NP)
  {
         # random integer between 1 and d
         rnbr = rand_int(1->d)
         # d floats between 0->1
         randb = rand_float(0->1, size=d)
         for (k=1 -> d)
         {
           if (k==rnbr) or (randb[k]<CR)
           {
              r1,r2,r3 = rand_ints(1->NP,
              all_different, != i) # three
              distinct rand int not equal to i
            # random vector plus a weighted
              differential
              u[i][k] = x[r1][k] + F*(x[r2][k]-
                                        x[r3][k])
            } else
            {
              u[i][k] = x[i][k]
            }
       # evaluate fitness of new solutions
       if f(u[i]) < f(x[i])
       {
         x[i] = u[i]
       }
       current_gen++
}
```

**Figure II. The DE algorithm in pseudocode**

The DE/rand/1/bin scheme was implemented in Python using the pseudocode provided in Storn and Price’s original formal paper on the matter [1]. Pseudocode is provided in Figure 2. No adaptation had to be made to the problem instances since DE is designed to be performed on real vector inputs.

## 5 EXPERIMENTS

### 5.1 Experiment Plan

**TABLE II.**
**Function evaluations per algorithm.**

| Algorithm | Func. Calls Formula | Allowed Iterations In Experiments |
|---|---|---|
| **DE** | NP+2*max_gen*NP | 333 |
| **SGD** | NP+(d+1)*max_gen*NP | 333 |
| **RandSrch** | 1+max_gen*NP | 666 |

The test problems used are shown in Table 1. DE was tested against a random search, as well as a stochastic hill climber by allowing for up to 20,000 function evaluations to converge to a result. Additionally, a population of 30 was used for each problem.

Each algorithm will use a different number of function evaluations for each iteration of the algorithm. Letting *max_gen* be the number of total iterations of population evolution in an algorithm, *NP* the size of the population, and *d* the number of dimensions of the problem, the details of each algorithm's function calls and allowed iterations are described in Table 2.

Clearly random search uses the least, and so for a set number of function calls can perform many more iterations of its algorithm. Stochastic gradient descent uses the most function calls, however this is due to the fact that exact derivatives were not available for each problem function and so a quickly written, inefficient, gradient approximation function was written. For this reason SGD was allowed the same number of iterations as DE.

### 5.2 Comparison Algorithms

The DE algorithm was compared against a basic random search function and a stochastic hill climber. These were implemented in Python. Random search is, as the name suggests, an algorithm that randomly searches the search space

### 5.3 Parameter tuning

**TABLE III.**
**Parameters for each test problem.**

| Problem | F | CR |
|---|---|---|
| **F1** | 0.11 | 0.15 |
| **F2** | 0.28 | 0.75 |
| **F3** | 0.28 | 0.95 |
| **F4** | 0.40 | 0.85 |
| **F5** | 0.28 | 0.70 |
| **F6** | 0.11 | 0.10 |
| **F7** | 0.11 | 0.35 |
| **F8** | 0.28 | 0.80 |
| **F9** | 0.28 | 0.95 |

There are many methods for optimising a model's hyperparameters, including Bayesian optimisation [Gaussian??] and using neural networks. Here, however, we perform a simple grid search over the domain CR ∈ (0,1] and F ∈ [0,1.2] to give a total of 420 parameter combinations.

The hyperparameter NP, the number of members of the population, was fixed at 20 in order to restrict CPU load and to more easily gauge the performance change of the other two parameters. The crossover rate, CR, and the differential factor, F, were both tuned with the grid search described above on various problem configurations using different optimisation problems and dimensions. Table 3 shows the best results for each problem formulation F1-9, and therefore the configurations used in the experiments.

### 5.4 Results

**TABLE IV.**
**Experimental Results.**

| Problem | Global Min. | DE | SGD | RandSrch |
|---|---|---|---|---|
| **F1** | -1 | -1 | -4.7e-165 | -0.655 |
| **F2** | 0 | 0.596 | 18.8 | 18.3 |
| **F3** | 0 | 0.469 | 19.1 | 19.9 |
| **F4** | 0 | 12.3 | 3.89e40 | 621 |
| **F5** | 0 | 53.5 | 7.82e44 | 3230 |
| **F6** | 0 | 2.83 | 81.9 | 133 |
| **F7** | 0 | 12.0 | 165 | 340 |
| **F8** | 0 | 0.124 | 234 | 110 |
| **F9** | 0 | 0.262 | 608 | 366 |

The results of the experiments are shown in Table 4, with the average score over ten iterations of each algorithm shown. Differential evolution far outperforms the other two algorithms in all cases, even converging perfectly for the first problem (Easom function). The SGD algorithm was not perfect and appeared to struggle on problems F4-5, instead diverging to huge values. Even so, the benefit of DE is apparent.

A simple two-dimensional problem such as used in F1 was easily solved by DE, with random search finding a good solution. SGD appeared to get stuck in local minima for some of the problems, and simply converging slowly for others.

## 6 CONCLUSION

This paper has presented an implementation of DE/rand/1/bin for minimising a range of objective functions, and compared the results against a stochastic gradient descent and a random search algorithm. The results clearly show the benefit of differential evolution, with impressive results in only a small number of generations (333) and a relatively small population (30). A better optimised implementation of this algorithm could likely perform even better with fewer function evaluations. The experiments were performed on various well established optimisation functions with a varying number of dimensions. Differential evolution can gain good results even on high-dimensional problems.

In future research, it would be valuable to compare also with variants such as DE/best/2 for example, and even implementing other novel variants of DE.

## REFERENCES

[1] Storn, R. and K. Price (1997). "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces." Journal of global optimization 11(4): 341.

[2] Storn, R. (1996). On the usage of differential evolution for function optimization. Proceedings of north american fuzzy information processing, Ieee.

[3] Storn, R. (1995). "Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical report." International Computer Science Institute 11.

[4] Fogel, Lawrence J. "Autonomous Automata." Industrial research 4 (1962): 14-19.

[5] Fogel, LJ. Extending Communication and Control through Simulated Evolution. Bioengineering—An Engineering View: Proceedings of the Symp. Engineering Significance of the Biological Sciences: San Francisco Press, San Francisco, 1968

[6] Rechenberg, Ingo. "Cybernetic Solution Path of an Experimental Problem." *Roy. Aircr. Establ., Libr. transl.* 1122 (1965).

[7] Schwefel, H-P. "Kybernetische Evolution Als Strategie Der Experimentellen Forschung in Der Stromungsmechanik." *Master's thesis, Technische Universitat Berlin, Hermann Fottinger Institut fuer Hydrodynamik* (1965).

[8] Rechenberg, Ingo. (1971) "Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien Der Biologischen Evolution. Dr.-Ing." Thesis, Technical University of Berlin, Department of Process Engineering.

[9] S. Das and P. N. Suganthan, (2010) "Differential Evolution: A Survey of the State-of-the-Art," in *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4-31, Feb. 2011, doi: 10.1109/TEVC.2010.2059031.

[10] Ackley, D. (1987). *A connectionist machine for genetic hillclimbing* (Vol. 28). Springer science & business media.

[11] Rastrigin, L. A. (1974). Systems of extremal control. *Nauka*.

[12] Rudolph, G. (1990). *Globale Optimierung mit parallelen Evolutionsstrategien* (Doctoral dissertation, Diplomarbeit, Universit at Dortmund, Fachbereich Informatik).

[13] Rosenbrock, H. H. "An Automatic Method for Finding the Greatest or Least Value of a Function." *The Computer Journal* 3, no. 3 (1960): 175-84

[14] Griewank, A. O. (1981). Generalized descent for global optimization. *Journal of optimization theory and applications*, *34*, 11-39.

[15] Chiang, C. W., Lee, W. P., & Heh, J. S. (2010). A 2-Opt based differential evolution for global optimization. *Applied Soft Computing*, *10*(4), 1200-1207.

[16] Das, S., Mullick, S. S., & Suganthan, P. N. (2016). Recent advances in differential evolution–an updated survey. *Swarm and evolutionary computation*, *27*, 1-30.