Differential Evolution*

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

April 2, 2010
Technical Report: CA-TR-20100402-1

Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Differential Evolution algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Differential Evolution`

1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent, and centralized manner [1]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [2]. This report describes the Differential Evolution algorithm using the standardized template.

2 Name

Differential Evolution, DE

3 Taxonomy

Differential Evolution is a Stochastic Direct Search and Global Optimization algorithm, and is an instance of an Evolutionary Algorithm from the field of Evolutionary Computation. It is related to sibling Evolutionary Algorithms such as the Genetic Algorithm and Evolutionary Programming, and Evolution Strategies, and shows some similarities to Particle Swarm Optimization.

4 Strategy

The Differential Evolution algorithm involves maintaining a population of candidate solutions subjected to iterations of recombination, evaluation, and selection. The recombination approach involves the creation of new candidate solution components based on the weighted difference between two randomly selected population members added to a third population member. This perturbs population members relative to the spread of the broader population. In conjunction with selection, the perturbation effect self-organizes the sampling of the problem space, bounding it to known areas of interest.

5 Procedure

Differential Evolution has a specialized nomenclature that describes the adopted configuration. This takes the form of DE/x/y/z, where x represents the solution to be perturbed (such a random or best). The y signifies the number of difference vectors used in the perturbation of x, where a difference vectors is the difference between two randomly selected although distinct members of the population. Finally, z signifies the recombination operator performed such as bin for binomial and exp for exponential.

Algorithm 1 provides a pseudo-code listing of the Differential Evolution algorithm for minimizing a cost function, specifically a DE/rand/1/bin configuration. Algorithm 2 provides a pseudo-code listing of the *NewSample* function from the Differential Evolution algorithm.

Algorithm 1: Pseudo Code for the Differential Evolution algorithm.

```
Input: G, NP, F, CR
Output: S_best
Population ← InitializePopulation(G, NP);
EvaluateCost(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    NewPopulation ← 0;
    foreach P_i ∈ Population do
        S_i ← NewSample(P_i, Population, NP, F, CR);
        if Cost(S_i) ≤ Cost(P_i) then
            NewPopulation ← S_i;
        else
            NewPopulation ← P_i;
        end
    end
    Population ← NewPopulation;
    EvaluateCost(Population);
    S_best ← GetBestSolution(Population);
end
return S_best;
```

6 Heuristics

- Differential evolution was designed for nonlinear, non-differentiable continuous function optimization.

Algorithm 2: Pseudo Code for the NewSample function in the Differential Evolution algorithm.

Input: P_0, Population, NP, F, CR
Output: S

repeat
  $P_1 \leftarrow$ `RandomMemeber(Population)`;
until $P_1 \neq P_0$;
repeat
  $P_2 \leftarrow$ `RandomMemeber(Population)`;
until $P_2 \neq P_0 \vee P_2 \neq P_1$;
repeat
  $P_3 \leftarrow$ `RandomMemeber(Population)`;
until $P_3 \neq P_0 \vee P_3 \neq P_1 \vee P_3 \neq P_2$;
CutPoint $\leftarrow$ `RandomPosition(NP)`;
$S \leftarrow 0$;
for i **to** NP **do**
  if $i \equiv$ CutPoint $\wedge$ `Rand()` $<$ CR **then**
    $S_i \leftarrow P3_i + \mathsf{F} \times (P1_i\text{-}\ P2_i)$;
  else
    $S_i \leftarrow P0_i$;
  end
end
return S;

- The weighting factor $F \in [0, 2]$ controls the amplification of differential variation, a value of 0.8 is suggested.
- the crossover weight $CR \in [0, 1]$ probabilistically controls the amount of recombination, a value of 0.9 is suggested.
- The initial population of candidate solutions should be randomly generated from within the space of valid solutions.
- The popular configurations are DE/rand/1/* and DE/best/2/*.

7 Code Listing

Listing 1 provides an example of the Differential Evolution algorithm implemented in the Ruby Programming Language. The demonstration problem is an instance of a continuous function optimization that seeks $minf(x)$ where $f = \sum_{i=1}^{n} x_i^2$, $-5.0 \leq x_i \leq 5.0$ and $n = 3$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$. The algorithm is an implementation of Differential Evolution with the DE/rand/1/bin configuration proposed by Storn and Price [11].

```ruby
def objective_function(vector)
  return vector.inject(0.0) {|sum, x| sum + (x ** 2.0)}
end

def random_vector(problem_size, search_space)
  return Array.new(problem_size) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
end

def new_sample(p0, p1, p2, p3, f, cr, search_space)
```

```ruby
  length = p0[:vector].length
  sample = {}
  sample[:vector] = []
  cut = rand(length-1) + 1
  length.times do |i|
    if (i==cut or rand() < cr)
      v = p3[:vector][i] + f * (p1[:vector][i] - p2[:vector][i])
      v = search_space[i][0] if v < search_space[i][0]
      v = search_space[i][1] if v > search_space[i][1]
      sample[:vector] << v
    else
      sample[:vector] << p0[:vector][i]
    end
  end
  return sample
end

def search(max_generations, np, search_space, g, f, cr)
  pop = Array.new(g) {|i| {:vector=>random_vector(np, search_space)} }
  pop.each{|c| c[:cost] = objective_function(c[:vector])}
  gen, best = 0, pop.sort{|x,y| x[:cost] <=> y[:cost]}.first
  max_generations.times do |gen|
    samples = []
    pop.each_with_index do |p0, i|
      p1 = p2 = p3 = -1
      p1 = rand(pop.length) until p1!=i
      p2 = rand(pop.length) until p2!=i and p2!=p1
      p3 = rand(pop.length) until p3!=i and p3!=p1 and p3!=p2
      samples << new_sample(p0, pop[p1], pop[p2], pop[p3], f, cr, search_space)
    end
    samples.each{|c| c[:cost] = objective_function(c[:vector])}
    nextgen = Array.new(g) do |i|
      (samples[i][:cost]<=pop[i][:cost]) ? samples[i] : pop[i]
    end
    pop = nextgen
    pop.sort{|x,y| x[:cost] <=> y[:cost]}
    best = pop.first if pop.first[:cost] < best[:cost]
    puts " > gen #{gen+1}, fitness=#{best[:cost]}"
  end
  return best
end


problem_size = 3
max_generations = 200
pop_size = 10*problem_size
weighting_factor = 0.8
crossover_factor = 0.9
search_space = Array.new(problem_size) {|i| [-5, +5]}

best = search(max_generations, problem_size, search_space, pop_size, weighting_factor,
    crossover_factor)
puts "done! Solution: f=#{best[:cost]}, s=#{best[:vector].inspect}"
```

Listing 1: Differential Evolution algorithm in the Ruby Programming Language

8 References

8.1 Primary Sources

The Differential Evolution algorithm was presented by Storn and Price in a technical report that considered DE1 and DE2 variants of the approach applied to a suite of continuous function

optimization problems [9]. An early paper by Storn applied the approach to the optimization of an IIR-filter (Infinite Impulse Response) [7]. A second early paper applied the approach to a second suite of benchmark problem instances, adopting the contemporary nomenclature for describing the approach, including the DE/rand/1 and DE/best/2 variations [10]. The early work including technical reports and conference papers by Storn and Price culminated in a seminal journal article [11].

8.2 Learn More

A classical overview of Differential Evolution is presented by Price and Storn [4], and terse introduction to the approach for function optimization is presented by Storn [8]. A seminal extended description of the algorithm with sample applications was presented by Storn and Price as a book chapter [5]. Price, Storn, and Lampinen release a contemporary book dedicated to Differential Evolution including theory, benchmarks, sample code and numerous application demonstrations [6]. Chakraborty also released a book considering extensions to the approach to address complexities such as rotation invariance and stopping criteria [3].

9 Conclusions

This report described the Differential Evolution algorithm as an instance of an Evolutionary Algorithm.

10 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is wrong by default. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at jasonb@CleverAlgorithms.com or visit the project website at http://www.CleverAlgorithms.com.

References

[1] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[2] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[3] Uday K. Chakraborty. *Advances in Differential Evolution*. Springer, 2008.

[4] K. Price and R. Storn. Differential evolution: Numerical optimization made easy. *Dr. Dobb's Journal*, pages 18–24, 1997.

[5] Kenneth V. Price. *New Ideas in Optimization*, chapter An introduction to differential evolution, pages 79–108. McGraw-Hill Ltd., UK, 1999.

[6] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer, 2005.

[7] R. Storn. Differential evolution design of an iir-filter. In *Proceedings IEEE Conference Evolutionary Computation*, pages 268–273. IEEE, 1996.

[8] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings Fuzzy Information Processing Society, 1996 Biennial Conference of the North American*, pages 519–523, 1996.

[9] R. Storn and K. Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[10] R. Storn and K. Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 842–844. IEEE, 1996.

[11] 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.