Evolutionary Programming*

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

March 13, 2010
Technical Report: CA-TR-20100313-1

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 Evolutionary Programming algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Evolutionary, Programming`

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 Evolutionary Programming algorithm using the standardized template.

2 Name

Evolutionary Programming, EP

3 Taxonomy

Evolutionary Programming is a Global Optimization algorithm and is an instance of an Evolutionary Algorithm from the field of Evolutionary Computation. The approach is a sibling of other Evolutionary Algorithms such as the Genetic Algorithm, and Learning Classifier Systems. It is sometimes confused with Genetic Programming given the similarity in name, and more recently it shows a strong functional similarity to Evolution Strategies.

4 Inspiration

Evolutionary Programming is inspired by the theory of evolution by means of natural selection. Specifically, the technique is inspired by macro-level or the species-level process of evolution (phenotype, hereditary, variation) and is not concerned with the genetic mechanisms of evolution (genome, chromosomes, genes, alleles).

5 Metaphor

A population of a species reproduce, creating progeny with small phenotypical variation. The progeny and the parents compete based on their suitability to the environment, where the generally more fit members constitute the subsequent generation and are provided with the opportunity to reproduce themselves. This process repeats, improving the adaptive fit between the species and the environment.

6 Strategy

The objective of the Evolutionary Programming algorithm is to maximize the suitability of collection of candidate solutions in the context of an objective function from the domain. This objective is pursued by using an adaptive model with surrogates for the processes of evolution, specifically hereditary (reproduction with variation) under competition. The representation used for candidate solutions is directly assessable by a cost or objective function from the domain, and credit assignment is directly apportioned to this representation.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the Evolutionary Programming algorithm for minimizing a cost function.

8 Heuristics

- The representation for candidate solutions should be domain specific, such as real numbers for continuous function optimization.
- The sample size (bout size) for tournament selection during competition is commonly between 5% and 10% of the population size.
- Evolutionary Programming traditionally only uses the mutation operator to create new candidate solutions from existing candidate solutions. The crossover operator that is used in some other Evolutionary Algorithms is not employed in Evolutionary Programming.
- Evolutionary Programming is concerned with the linkage between parent and child candidate solutions and is not concerned with surrogates for genetic mechanisms.
- Continuous function optimization is a popular application for the approach, where real-valued representations are with a Gaussian-based mutation operator.
- The mutation-specific parameters used in the application of the algorithm to continuous function optimization can be adapted in concert with the candidate solutions [5].

Algorithm 1: Pseudo Code for the Evolutionary Programming algorithm.

Input: $Population_{size}$, ProblemSize, BoutSize
Output: S_{best}

```
Population ← InitializePopulation(Population_size, ProblemSize);
EvaluatePopulation(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    Children ← 0;
    foreach Parent_i ∈ Population do
        Child_i ← Mutate(Parent_i);
        Children ← Child_i;
    end
    EvaluatePopulation(Children);
    S_best ← GetBestSolution(Children, S_best);
    Union ← Population + Children;
    foreach S_i ∈ Union do
        for 1 to BoutSize do
            S_j ← RandomSelection(Union);
            if Cost(S_i) < Cost(S_j) then
                Si_wins ← Si_wins + 1;
            end
        end
    end
    Population ← SelectBestByWins(Union, Population_size);
end
return S_best;
```

9 Code Listing

Listing 1 provides an example of the Evolutionary Programming 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 $n = 2$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$. The algorithm is an implementation of Evolutionary Programming based on Fogel et al's classical implementation for continuous function optimization [5] with per-variable adaptive variance based on Fogel's description for a self-adaptive variation on page 160 of his 1995 book [6].

```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 random_gaussian
  u1 = u2 = w = g1 = g2 = 0
  begin
    u1 = 2 * rand() - 1
    u2 = 2 * rand() - 1
    w = u1 * u1 + u2 * u2
  end while w >= 1
```

```
  w = Math::sqrt((-2 * Math::log(w)) / w)
  g2 = u1 * w;
  g1 = u2 * w;
  return g1
end

def mutate(candidate, search_space)
  child = {}
  child[:vector], child[:strategy] = [], []
  candidate[:vector].each_with_index do |v_old, i|
    s_old = candidate[:strategy][i]
    v = v_old + s_old * random_gaussian()
    v = search_space[i][0] if v < search_space[i][0]
    v = search_space[i][1] if v > search_space[i][1]
    child[:vector] << v
    s = s_old + ((s_old<0) ? s_old*-1.0 : s_old)**0.5 * random_gaussian()
    child[:strategy] << s
  end
  return child
end

def distance(v1, v2)
  sum = 0.0
  v1.each_with_index {|v,i| sum += (v-v2[i])**2.0}
  return Math::sqrt(sum)
end

def tournament(candidate, population, bout_size)
  candidate[:wins] = 0
  bout_size.times do |i|
    other = population[rand(population.length)]
    candidate[:wins] += 1 if candidate[:fitness] < other[:fitness]
  end
end

def search(max_generations, problem_size, search_space, pop_size, bout_size)
  strategy_space = Array.new(problem_size) do |i|
    [0, (search_space[i][1]-search_space[i][0])*0.02]
  end
  population = Array.new(pop_size) do |i|
    {:vector=>random_vector(problem_size, search_space),
      :strategy=>random_vector(problem_size, strategy_space)}
  end
  population.each{|c| c[:fitness] = objective_function(c[:vector])}
  gen, best = 0, population.sort{|x,y| x[:fitness] <=> y[:fitness]}.first
  max_generations.times do |gen|
    children = Array.new(pop_size) {|i| mutate(population[i], search_space)}
    children.each{|c| c[:fitness] = objective_function(c[:vector])}
    children.sort!{|x,y| x[:fitness] <=> y[:fitness]}
    best = children.first if children.first[:fitness] < best[:fitness]
    union = children+population
    union.each{|c| tournament(c, union, bout_size)}
    union.sort!{|x,y| y[:wins] <=> x[:wins]}
    population = union[0...pop_size]
    puts " > gen #{gen}, fitness=#{best[:fitness]}"
  end
  return best
end

max_generations = 200
population_size = 100
problem_size = 2
search_space = Array.new(problem_size) {|i| [-5, +5]}
```

```
81 bout_size = 5
82
83 best = search(max_generations, problem_size, search_space, population_size, bout_size)
84 puts "done! Solution: f=#{best[:fitness]}, s=#{best[:vector].inspect}"
```

Listing 1: Evolutionary Programming algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

Evolutionary Programming was developed by Lawrence Fogel, outlined in early papers (such as [7]) and later became the focus of his PhD dissertation [8]. Fogel focused on the use of an evolutionary process for the development of control systems using Finite State Machine (FSM) representations. Fogel's early work on Evolutionary Programming culminated in a book, co-authored with Owens and Walsh that elaborated the approach, focusing on the evolution of state machines for the prediction of symbols in time series data [11].

10.2 Learn More

The field of Evolutionary Programming lay relatively dormant for 30 years until it was revived by Fogel's son, David Fogel. Early works considered the application of Evolutionary Programming to control systems [13], and later function optimization (system identification) culminating in a book on the approach [3], and David Fogel's PhD dissertation [4]. Lawrence Fogel collaborated in the revival of the technique, including reviews [9, 10] and extensions on what became the focus of the approach on function optimization [5].

Yao, et al. provide a seminal study of Evolutionary Programming proposing an extension and racing it against the classical approach on a large number of test problems [14]. Finally, Porto provides an excellent contemporary overview of the field and the technique [12].

11 Conclusions

This report described the Evolutionary Programming algorithm as a classical Evolutionary Algorithm that has been revived for application to continuous function optimization.

12 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] D. B. Fogel. *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Needham Heights, 1991.

[4] D. B. Fogel. *Evolving artificial intelligence*. PhD thesis, University of California, San Diego, CA, USA, 1992.

[5] D. B. Fogel, L. J. Fogel, and J.W. Atmar. Meta-evolutionary programming. In *Proc. 25th Asilomar Conf. Signals, Systems, and Computers*, pages 540–545, 1991.

[6] David B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*. IEEE Press, 1995.

[7] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[8] L. J. Fogel. *On the Organization of Intellect*. PhD thesis, UCLA, 1964.

[9] L. J. Fogel. The future of evolutionary programming. In *Proceedings of the Conference on Signals, Systems and Computers*, 1990.

[10] L. J. Fogel. *Computational Intelligence: Imitating Life*, chapter Evolutionary Programming in Perspective: the Top-down View, pages 135–146. IEEE Press, 1994.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[12] V. William Porto. *Evolutionary Computation 1: Basic Algorithms and Operations*, chapter 10: Evolutionary Programming, pages 89–102. IoP Press, 2000.

[13] A.V. Sebald and D.B. Fogel. Design of SLAYR neural networks using evolutionary programming. In *Proceedings. of the 24th Asilomar Conf. on Signals, Systems and Computers*, pages 1020–1024, 1990.

[14] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.