Memetic Algorithm*

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

November 16, 2010
Technical Report: CA-TR-20101116-1

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 Memetic Algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Memetic, Meme`

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 Memetic Algorithm using the standardized template.

2 Name

Memetic Algorithm, MA

3 Taxonomy

Memetic Algorithms have elements of Metaheuristics and Computational Intelligence. Although they have principles of Evolutionary Algorithms, they may not strictly be considered an Evolutionary Technique. Memetic Algorithms have functional similarities to Baldwinian Evolutionary Algorithms, Lamarckian Evolutionary Algorithms, Hybrid Evolutionary Algorithms, and Cultural Algorithms. Using ideas of memes and Memetic Algorithms in optimization may be referred to as Memetic Computing.

4 Inspiration

Memetic Algorithms are inspired by the interplay of genetic evolution and memetic evolution. Universal Darwinism is the generalization of genes beyond biological-based systems to any system where discrete units of information can be inherited and be subjected to evolutionary forces of selection and variation. The term 'meme' is used to refer to a piece of discrete cultural information, suggesting at the interplay of genetic and cultural evolution.

5 Metaphor

The genotype is evolved based on the interaction the phenotype has with the environment. This interaction is metered by cultural phenomena that influence the selection mechanisms, and even the pairing and recombination mechanisms. Cultural information is shared between individuals, spreading through the population as memes relative to their fitness or fitness the memes impart to the individuals. Collectively, the interplay of the geneotype and the memeotype strengthen the fitness of population in the environment.

6 Strategy

The objective of the information processing strategy is to exploit a population based global search technique to broadly locate good areas of the search space, combined with the repeated usage of a local search heuristic by individual solutions to locate local optimum. Ideally, memetic algorithms embrace the duality of genetic and cultural evolution, allowing the transmission, selection, inheritance, and variation of memes as well as genes.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the Memetic Algorithm for minimizing a cost function. The procedure describes a simple or first order Memetic Algorithm that shows the improvement of individual solutions separate from a global search, although does not show the independent evolution of memes.

Algorithm 1: Pseudo Code for the Memetic Algorithm.

Input: ProblemSize, Pop_{size}, $MemePop_{size}$
Output: S_{best}

```
Population ← InitializePopulation(ProblemSize, Pop_size);
while ¬StopCondition() do
    foreach S_i ∈ Population do
        Si_cost ← Cost(S_i);
    end
    S_best ← GetBestSolution(Population);
    Population ← StochasticGlobalSearch(Population);
    MemeticPopulation ← SelectMemeticPopulation(Population, MemePop_size);
    foreach S_i ∈ MemeticPopulation do
        S_i ← LocalSearch(S_i);
    end
end
return S_best;
```

8 Heuristics

- The global search provides the broad exploration mechanism, whereas the individual solution improvement via local search provides an exploitation mechanism.
- Balance is needed between the local and global mechanisms to ensure the system does not prematurely converge to a local optimum and does not consume unnecessary computational resources.
- The local search should be problem and representation specific, where as the global search may be generic and non-specific (black-box).
- Memetic Algorithms have been applied to a range of constraint, combinatorial, and continuous function optimization problem domains.

9 Code Listing

Listing 1 provides an example of the Memetic 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 Memetic Algorithm uses a canonical Genetic Algorithm as the global search technique that operates on binary strings, uses tournament selection, point mutations, uniform crossover and a binary coded decimal decoding of bits to real values. A bit climber local search is used that performs probabilistic bit flips (point mutations) and only accepts solutions with the same or improving fitness.

```
BITS_PER_PARAM = 16

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

def random_bitstring(num_bits)
  return (0...num_bits).inject(""){|s,i| s<<((rand<0.5) ? "1" : "0")}
end

def decode(bitstring, search_space)
  vector = []
  search_space.each_with_index do |bounds, i|
    off, sum, j = i*BITS_PER_PARAM, 0.0, 0
    bitstring[off...(off+BITS_PER_PARAM)].reverse.each_char do |c|
      sum += ((c=='1') ? 1.0 : 0.0) * (2.0 ** j.to_f)
      j += 1
    end
    min, max = bounds
    vector << min + ((max-min)/((2.0**BITS_PER_PARAM.to_f)-1.0)) * sum
  end
  return vector
end

def fitness(candidate, search_space)
  candidate[:vector] = decode(candidate[:bitstring], search_space)
  candidate[:fitness] = objective_function(candidate[:vector])
end

def binary_tournament(population)
  s1, s2 = population[rand(population.size)], population[rand(population.size)]
  return (s1[:fitness] > s2[:fitness]) ? s1 : s2
end
```

```ruby

def point_mutation(bitstring, prob_mutation)
  string = ""
   bitstring.each_char do |bit|
    string << ((rand()<prob_mutation) ? ((bit=='1') ? "0" : "1") : bit)
  end
  return string
end

def uniform_crossover(parent1, parent2, p_crossover)
  return ""+parent1 if rand()>=p_crossover
  child = ""
  parent1.length.times do |i|
    child << ((rand()<0.5) ? parent1[i].chr : parent2[i].chr)
  end
  return child
end

def reproduce(selected, population_size, p_crossover, p_mutation)
  children = []
  selected.each_with_index do |p1, i|
    p2 = (i.even?) ? selected[i+1] : selected[i-1]
    child = {}
    child[:bitstring] = uniform_crossover(p1[:bitstring], p2[:bitstring], p_crossover)
    child[:bitstring] = point_mutation(child[:bitstring], p_mutation)
    children << child
    break if children.size >= population_size
  end
  return children
end

def bitclimber(child, search_space, p_mutation, max_local_gens)
  current = child
  max_local_gens.times do
    candidate = {}
    candidate[:bitstring] = point_mutation(current[:bitstring], p_mutation)
    fitness(candidate, search_space)
    current = candidate if candidate[:fitness] <= current[:fitness]
  end
  return current
end

def search(max_gens, problem_size, search_space, pop_size, p_crossover, p_mutation,
    max_local_gens, p_local)
  pop = Array.new(pop_size) do |i|
    {:bitstring=>random_bitstring(problem_size*BITS_PER_PARAM)}
  end
  pop.each{|candidate| fitness(candidate, search_space) }
  gen, best = 0, pop.sort{|x,y| x[:fitness] <=> y[:fitness]}.first
  max_gens.times do |gen|
    selected = Array.new(pop_size){|i| binary_tournament(pop)}
    children = reproduce(selected, pop_size, p_crossover, p_mutation)
    children.each{|candidate| fitness(candidate, search_space) }
    pop = []
    children.each do |child|
      child = bitclimber(child, search_space, p_mutation, max_local_gens) if rand() < p_local
      pop << child
    end
    pop.sort!{|x,y| x[:fitness] <=> y[:fitness]}
    best = pop.first if pop.first[:fitness] <= best[:fitness]
    puts ">gen=#{gen}, f=#{best[:fitness]}, b=#{best[:bitstring]}, v=#{best[:vector].inspect}"
  end
  return best
```

```ruby
end

if __FILE__ == $0
  problem_size = 3
  search_space = Array.new(problem_size) {|i| [-5, +5]}
  max_gens = 100
  pop_size = 100
  p_crossover = 0.98
  p_mutation = 1.0/(problem_size*BITS_PER_PARAM).to_f
  max_local_gens = 20
  p_local = 0.5

  best = search(max_gens, problem_size, search_space, pop_size, p_crossover, p_mutation,
      max_local_gens, p_local)
  puts "done! Solution: f=#{best[:fitness]}, b=#{best[:bitstring]}, v=#{best[:vector].inspect}"
end
```

Listing 1: Memetic Algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

The concept of a Memetic Algorithm is credited to Moscato [8], who was inspired by the description of meme's in Dawkins' "The Selfish Gene" [3]. Moscato proposed Memetic Algorithms as the marriage between population based global search and heuristic local search made by each individual without the constraints of a genetic representation and investigated variations on the Traveling Salesman Problem.

10.2 Learn More

Moscato and Cotta provide a gentle introduction to the field of Memetic Algorithms as a book chapter that covers formal descriptions of the approach, a summary of the fields of application, and the state of the art [7]. A overview and classification of the types of Memetic Algorithms is presented by Ong, et al. who describe a class of adaptive Memetic Algorithms [9]. Krasnogor and Smith also provide a taxonomy of Memetic Algorithms, focusing on the properties needed to design 'competent' implementations of the approach with examples on a number of combinatorial optimization problems [6]. Work by Krasnogor and Gustafson investigate what they refer to as 'self-generating' Memetic Algorithms that use the memetic principle to co-evolve the local search applied by individual solutions [5]. For a broader overview of the field, see the 2005 book "Recent Advances in Memetic Algorithms" that provides an overview and a number of studies [4].

11 Conclusions

This report described the Memetic algorithm using the standardized algorithm template.

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 subjected to continuous improvement. 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] Richard Dawkins. *The selfish gene*. Oxford University Press, 1976.

[4] William E. Hart, Natalio Krasnogor, and J.E. Smith. *Recent Advances in Memetic Algorithms*. Springer, 2005.

[5] N. Krasnogor and S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.

[6] Natalio Krasnogor and Jim Smith. A tutorial for competent memetic algorithms: Model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[7] P. Moscato and C. Cotta. *Handbook of Metaheuristics*, chapter A gentle introduction to memetic algorithms, pages 105–144. Kluwer Academic Publishers, 2003.

[8] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, California Institute of Technology, 1989.

[9] Yew-Soon Ong, Meng-Hiot Lim, Ning Zhu, and Kok-Wai Wong. Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 36(1):141–152, 2006.