Compact Genetic Algorithm*

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

November 18, 2010
Technical Report: CA-TR-20101118c-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 Compact Genetic Algorithm using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Compact, Genetic, Algorithm`

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 Compact Genetic Algorithm using the standardized algorithm description template.

2 Name

Compact Genetic Algorithm, CGA, cGA

3 Taxonomy

The Compact Genetic Algorithm is an Estimation of Distribution Algorithm (EDA), also referred to as Population Model-Building Genetic Algorithms (PMBGA), an extension to the field of Evolutionary Computation. The Compact Genetic Algorithm is the basis for extensions such as the Extended Compact Genetic Algorithm (ECGA). It is related to other EDAs such as the Univariate Marginal Probability Algorithm, the Population-Based Incremental Learning algorithm, and the Bayesian Optimization Algorithm.

4 Inspiration

The Compact Genetic Algorithm is a probabilistic technique without an inspiration. It is related to the Genetic Algorithm and other Evolutionary algorithms that are inspired by the biological theory of evolution by means of natural selection.

5 Strategy

The information processing objective of the algorithm is to simulate the behavior of a Genetic Algorithm with a much smaller memory footprint (without requiring a population to be maintained). This is achieved by maintaining a vector that specifies the probability of including each component in a solution in new candidate solutions. Candidate solutions are probabilistically generated from the vector and the components in the better solution are used to make small changes to the probabilities in the vector.

6 Procedure

The Compact Genetic Algorithm maintains a real-valued prototype vector that represents the probability of each component being expressed in a candidate solution. Algorithm 1 provides a pseudo-code listing of the Compact Genetic Algorithm for maximizing a cost function. The parameter n indicates the amount to update probabilities for conflicting bits in each algorithm iteration.

Algorithm 1: Pseudo Code for the Compact Genetic Algorithm.

Input: $Bits_{num}$, n
Output: S_{best}
$V \leftarrow$ `InitializeVector(`$Bits_{num}$, *0.5*`)`;
$S_{best} \leftarrow 0$;
while ¬`StopCondition()` **do**
  $S_1 \leftarrow$ `GenerateSamples(`V`)`;
  $S_2 \leftarrow$ `GenerateSamples(`V`)`;
  S_{winner}, $S_{loser} \leftarrow$ `SelectWinnerAndLoser(`S_1, S_2`)`;
  if `Cost(`S_{winner}`)` $\leq$ `Cost(`S_{best}`)` **then**
    $S_{best} \leftarrow S_{winner}$;
  end
  for i **to** $Bits_{num}$ **do**
    if $S^i_{winner} \neq S^i_{loser}$ **then**
      if $S^i_{winner} \equiv 1$ **then**
        $V^i_i \leftarrow V^i_i + \frac{1}{n}$;
      else
        $V^i_i \leftarrow V^i_i - \frac{1}{n}$;
      end
    end
  end
end
return S_{best};

7 Heuristics

- The vector update parameter (n) influences the amount that the probabilities are updated each algorithm iteration.
- The vector update parameter (n) may be considered to be comparable to the population size parameter in the Genetic Algorithm.
- Initial results demonstrate that the cGA may be comparable to a standard Genetic Algorithm on classical binary string optimization problems (such as onemax).
- The algorithm may be considered to have converged if the vector probabilities are all > 0 or < 1

8 Code Listing

Listing 1 provides an example of the Compact Genetic Algorithm implemented in the Ruby Programming Language. The demonstration problem is a maximizing binary optimization problem called OneMax that seeks a binary string of unity (all '1' bits). The objective function provides only an indication of the number of correct bits in a candidate string, not the positions of the correct bits. The algorithm is an implementation of Compact Genetic Algorithm that uses integer values to represent 1 and 0 nits in a binary string representation.

```
def onemax(vector)
  return vector.inject(0){|sum, value| sum + value}
end

def generate_candidate(vector)
  candidate = {}
  candidate[:bitstring] = Array.new(vector.length)
  vector.each_with_index do |p, i|
    candidate[:bitstring][i] = (rand()<p) ? 1 : 0
  end
  candidate[:cost] = onemax(candidate[:bitstring])
  return candidate
end

def update_vector(vector, winner, loser, population_size)
  vector.length.times do |i|
    if winner[:bitstring][i] != loser[:bitstring][i]
      if winner[:bitstring][i] == 1
        vector[i] = vector[i] + 1.0/population_size.to_f
      else
        vector[i] = vector[i] - 1.0/population_size.to_f
      end
    end
  end
end

def search(num_bits, max_iterations, population_size)
  vector = Array.new(num_bits){0.5}
  best = nil
  max_iterations.times do |iter|
    c1 = generate_candidate(vector)
    c2 = generate_candidate(vector)
    winner, loser = (c1[:cost] > c2[:cost] ? [c1,c2] : [c2,c1])
    best = winner if best.nil? or winner[:cost]>best[:cost]
    update_vector(vector, winner, loser, population_size)
    puts " >iteration=#{iter}, f=#{best[:cost]}, s=#{best[:bitstring]}"
  end
```

```
  return best
end

if __FILE__ == $0
  num_bits = 32
  max_iterations = 130
  population_size = 20

  best = search(num_bits, max_iterations, population_size)
  puts "done! Solution: f=#{best[:cost]}/#{num_bits}, s=#{best[:bitstring]}"
end
```

Listing 1: Compact Genetic Algorithm in the Ruby Programming Language

9 References

9.1 Primary Sources

The Compact Genetic Algorithm was proposed by Harik, Lobo, and Goldberg in 1999 [4], based on a random walk model perviously introduced by Harik, et al. [3]. In the introductory paper, the cGA is demonstrated to be comparable to the Genetic Algorithm on standard binary string optimization problems.

9.2 Learn More

Harik, et al. extend the Compact Genetic Algorithm (called the Extended Compact Genetic Algorithm) to generate populations of candidate solutions and perform selection (much like the Univariate Marginal Probabilist Algorithm), although it uses Marginal Product Models [5, 6]. Sastry and Goldberg performed further analysis into the Extended Compact Genetic Algorithm applying the method to a complex optimization problem [7].

10 Conclusions

This report described the Compact Genetic Algorithm using the standardized algorithm description template.

11 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] G. Harik, E. Cantu-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *IEEE International Conference on Evolutionary Computation*, pages 7–12, 1997.

[4] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

[5] Georges Harik. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). Technical Report 99010, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, 1999.

[6] Georges R. Harik, Fernando G. Lobo, and Kumara Sastry. *Scalable Optimization via Probabilistic Modeling*, chapter Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA), pages 39–61. Springer, 2006.

[7] K. Sastry and D. E. Goldberg. On extended compact genetic algorithm. In *Late Breaking Paper in Genetic and Evolutionary Computation Conference*, pages 352–359, 2000.