5.4 Compact Genetic Algorithm

Compact Genetic Algorithm, CGA, cGA.

5.4.1 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 (Section 5.3), the Population-Based Incremental Learning algorithm (Section 5.2), and the Bayesian Optimization Algorithm (Section 5.5).

5.4.2 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.

5.4.3 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.

5.4.4 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 5.4.1 provides a pseudocode 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.

5.4.5 Heuristics

- The vector update parameter (n) influences the amount that the probabilities are updated each algorithm iteration.

Algorithm 5.4.1: Pseudocode for the cGA.

```
Input: $Bits_{num}$, $n$
Output: $S_{best}$
1  $V \leftarrow$ InitializeVector($Bits_{num}$, $0.5$);
2  $S_{best} \leftarrow \emptyset$;
3  while $\neg$StopCondition() do
4      $S_1 \leftarrow$ GenerateSamples($V$);
5      $S_2 \leftarrow$ GenerateSamples($V$);
6      $S_{winner}$, $S_{loser} \leftarrow$ SelectWinnerAndLoser($S_1$, $S_2$);
7      if Cost($S_{winner}$) $\leq$ Cost($S_{best}$) then
8          $S_{best} \leftarrow S_{winner}$;
9      end
10     for $i$ to $Bits_{num}$ do
11         if $S^i_{winner} \neq S^i_{loser}$ then
12             if $S^i_{winner} \equiv 1$ then
13                 $V^i_i \leftarrow V^i_i + \frac{1}{n}$;
14             else
15                 $V^i_i \leftarrow V^i_i - \frac{1}{n}$;
16             end
17         end
18     end
19 end
20 return $S_{best}$;
```

- The vector update parameter (n) may be considered to be comparable to the population size parameter in the Genetic Algorithm.
- Early 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 either 0 or 1.

5.4.6 Code Listing

Listing 5.3 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 only provides 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 bits in a binary string representation.

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

def generate_candidate(vector)
  candidate = {}
  candidate[:bitstring] = Array.new(vector.size)
  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, pop_size)
  vector.size.times do |i|
    if winner[:bitstring][i] != loser[:bitstring][i]
      if winner[:bitstring][i] == 1
        vector[i] += 1.0/pop_size.to_f
      else
        vector[i] -= 1.0/pop_size.to_f
      end
    end
  end
end

def search(num_bits, max_iterations, pop_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, pop_size)
    puts " >iteration=#{iter}, f=#{best[:cost]}, s=#{best[:bitstring]}"
    break if best[:cost] == num_bits
  end
  return best
end

if __FILE__ == $0
  # problem configuration
  num_bits = 32
  # algorithm configuration
  max_iterations = 200
  pop_size = 20
  # execute the algorithm
  best = search(num_bits, max_iterations, pop_size)
  puts "done! Solution: f=#{best[:cost]}/#{num_bits},
       s=#{best[:bitstring]}"
end
```

Listing 5.3: Compact Genetic Algorithm in Ruby

5.4.7 References

Primary Sources

The Compact Genetic Algorithm was proposed by Harik, Lobo, and Goldberg in 1999 [3], based on a random walk model previously introduced by Harik et al. [2]. In the introductory paper, the cGA is demonstrated to be comparable to the Genetic Algorithm on standard binary string optimization problems.

Learn More

Harik et al. extended 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 used Marginal Product Models [1, 4]. Sastry and Goldberg performed further analysis into the Extended Compact Genetic Algorithm applying the method to a complex optimization problem [5].

5.4.8 Bibliography

[1] G. R. 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.

[2] G. R. Harik, E. Cantú-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.

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

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

[5] 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.