Univariate Marginal Distribution Algorithm*

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

November 22, 2010
Technical Report: CA-TR-20101118b-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 Univariate Marginal Distribution Algorithm using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Univariate, Marginal, Distribution, 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 [2]. 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 [3]. This report describes the Univariate Marginal Distribution Algorithm using the standardized algorithm description template.

2 Name

Univariate Marginal Distribution Algorithm, UMDA, Univariate Marginal Distribution, UMD

3 Taxonomy

The Univariate Marginal Distribution Algorithm belongs to the field of Estimation of Distribution Algorithms (EDA), also referred to as Population Model-Building Genetic Algorithms (PMBGA) an extension to the field of Evolutionary Computation. UMDA is closely related to the Factorized Distribution Algorithm (FDA) and an extension called the Bivariate Marginal Distribution Algorithm (BMDA). UMDA is related to other EDAs such as the Compact Genetic

Algorithm, the Population-Based Incremental Learning algorithm, and the Bayesian Optimization Algorithm.

4 Inspiration

Population-Based Incremental Learning is a Population 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 strategy of the algorithm is to use the frequency of the components in a population of candidate solutions in the construction of new candidate solutions. This is achieved by first measuring the frequency of each component in the population (the univariate marginal probability) and using the probabilities to influence the probabilistic selection of components in the component-wise construction of new candidate solutions.

6 Procedure

Algorithm 1 provides a pseudo-code listing of the Univariate Marginal Distribution Algorithm for minimizing a cost function.

Algorithm 1: Pseudo Code for the Univariate Marginal Distribution Algorithm.

```
Input: Bits_num, Population_size, Selection_size
Output: S_best
Population ← InitializePopulation(Bits_num, Population_size);
EvaluatePopulation(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    Selected ← SelectFitSolutions(Population);
    V ← CalculateFrequencyOfComponents(Selected);
    OffSpring ← 0;
    for i to Population_size do
        OffSpring ← ProbabilisticallyConstructSolution(V);
    end
    EvaluatePopulation(OffSpring);
    S_best ← GetBestSolution(OffSpring);
    Population ← OffSpring;
end
return S_best;
```

7 Heuristics

- UMDA was designed for problems where the components of a solution are independent (linearly separable).
- A selection method is needed to identify the subset of good solutions from which to calculate the univariate marginal probabilities. Many selection methods from the field of Evolutionary Computation may be used.

8 Code Listing

Listing 1 provides an example of the Univariate Marginal Distribution 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 UMDA that uses the integers 1 and 0 to represent bits in a binary string representation. A binary tournament selection strategy is used and the whole population is replaced each iteration. The mechanisms from Evolutionary Computation such as elitism and more elaborate selection methods may be implemented in an extension.

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

def random_bitstring(length)
  return Array.new(length){ ((rand()<0.5) ? 1 : 0) }
end

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

def calculate_bit_probabilities(num_bits, pop)
  vector= Array.new(num_bits, 0.0)
  pop.each do |member|
    member[:bitstring].each_with_index {|v, i| vector[i] += v}
  end
  vector.each_with_index {|f,i| vector[i] = (f.to_f/pop.length.to_f)}
  return vector
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
  return candidate
end

def search(num_bits, max_iterations, population_size, selection_size)
  pop = Array.new(population_size) do
    {:bitstring=>random_bitstring(num_bits)}
  end
  pop.each{|c| c[:fitness] = onemax(c[:bitstring])}
  best = pop.sort{|x,y| y[:fitness] <=> x[:fitness]}.first
  max_iterations.times do |iter|
    selected = Array.new(selection_size) { binary_tournament(pop) }
    vector = calculate_bit_probabilities(num_bits, selected)
    samples = Array.new(population_size) { generate_candidate(vector) }
    samples.each{|c| c[:fitness] = onemax(c[:bitstring])}
    samples.sort{|x,y| y[:fitness] <=> x[:fitness]}
    best = samples.first if samples.first[:fitness] > best[:fitness]
    pop = samples
    puts " >iteration=#{iter}, f=#{best[:fitness]}, s=#{best[:bitstring]}"
  end
  return best
end

```

```
if __FILE__ == $0
  num_bits = 64
  max_iterations = 100
  population_size = 50
  selection_size = 30

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

Listing 1: Univariate Marginal Distribution Algorithm in the Ruby Programming Language

9 References

9.1 Primary Sources

The Univariate Marginal Distribution Algorithm was described by Mühlenbein in 1997 in which a theoretical foundation is provided (for the field of investigation in general and the algorithm specifically) [4]. Interestingly, Mühlenbein also describes and incremental version of UMDA (IUMDA) that is described as being equivalent to Baluja's Population-Based Incremental Learning (PBIL) algorithm [1].

9.2 Learn More

Pelikan and Mühlenbein extended the approach to cover problems that have dependencies between the components (specifically pair-dependencies), referring to the technique as the Bivariate Marginal Distribution Algorithm (BMDA) [5, 6].

10 Conclusions

This report described the Univariate Marginal Distribution 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] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, June 1994.

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

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

[4] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[5] Martin Pelikan and Heinz Muehlenbein. Marginal distributions in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel*, 1998.

[6] Martin Pelikan and Hans Mühlenbein. *Advances in Soft Computing: Engineering Design and Manufacturing*, chapter The Bivariate Marginal Distribution Algorithm, pages 521–535. Springer, 1999.