Population-Based Incremental Learning*

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

November 18, 2010
Technical Report: CA-TR-20101118a-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 Population-Based Incremental Learning algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Population-Based, Incremental, Learning`

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 [5]. 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 [6]. This report describes the Population-Based Incremental Learning algorithm using the standardized template.

2 Name

Population-Based Incremental Learning, PBIL

3 Taxonomy

Population-Based Incremental Learning 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. PBIL is related to other EDAs such as the Compact Genetic Algorithm, the Probabilistic Incremental Programing Evolution Algorithm, and the Bayesian Optimization Algorithm. The fact the the algorithm maintains a single prototype vector that is updated competitively shows some relationship to the Learning Vector Quantization 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 objective of the PBIL algorithm is to reduce the memory required by the genetic algorithm. This is done by reducing the population of a candidate solutions to a single prototype vector of attributes from which candidate solutions can be generated and assessed. Updates and mutation operators are also performed to the prototype vector, rather than the generated candidate solutions.

6 Procedure

The Population-Based Incremental Learning algorithm maintains on 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 Population-Based Incremental Learning algorithm for minimizing a cost function.

Algorithm 1: Pseudo Code for the Population-Based Incremental Learning.

Input: $Bits_{num}$, $Samples_{num}$, $Learn_{rate}$, $P_{mutation}$, $Mutation_{factor}$
Output: S_{best}
$V \leftarrow$ `InitializeVector(`$Bits_{num}$`)`;
$S_{best} \leftarrow 0$;
while ¬`StopCondition()` **do**
  $S_{current} \leftarrow 0$;
  for i **to** $Samples_{num}$ **do**
    $S_i \leftarrow$ `GenerateSamples(`V`)`;
    if `Cost(`S_i`)` $\leq$ `Cost(`$S_{current}$`)` **then**
      $S_{current} \leftarrow S_i$;
      if `Cost(`S_i`)` $\leq$ `Cost(`S_{best}`)` **then**
        $S_{best} \leftarrow S_i$;
      end
    end
  end
  foreach $S^i_{bit} \in S_{current}$ **do**
    $V^i_{bit} \leftarrow V^i_{bit} \times (1.0\text{-}Learn_{rate}) + S^i_{bit} \times Learn_{rate}$;
    if `Rand()` $< P_{mutation}$ **then**
      $V^i_{bit} \leftarrow V^i_{bit} \times (1.0\text{-}Mutation_{factor})$ + `Rand()` $\times Mutation_{factor}$;
    end
  end
end
return S_{best};

7 Heuristics

- PBIL was designed to optimize the probability of components from low cardinality sets, such as bit's in a binary string.
- The algorithm has a very small memory footprint (compared to some population-based evolutionary algorithms) given the compression of information into a single prototype vector.
- Extensions to PBIL have been proposed that extend the representation beyond sets to real-valued vectors.
- Variants of PBIL that were proposed in the original paper include updating the prototype vector with more than one competitive candidate solution (such as an average of top candidate solutions), and moving the prototype vector away from the least competitive candidate solution each iteration.
- Low learning rates are preferred, such as 0.1.

8 Code Listing

Listing 1 provides an example of the Population-Based Incremental Learning 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 the simple PBIL algorithm that updates the prototype vector based on the best candidate solution generated each iteration.

```
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
  return candidate
end

def update_vector(vector, current, lrate)
  vector.each_with_index do |p, i|
    vector[i] = p * (1.0-lrate) + current[:bitstring][i] * lrate
  end
end

def mutate_vector(vector, current, p_mutate, mutate_factor)
  vector.each_with_index do |p, i|
    if rand() < p_mutate
      vector[i] = p * (1.0-mutate_factor) + rand() * mutate_factor
    end
  end
end

def search(num_bits, max_iterations, num_samples, p_mutate, mutate_factor, learn_rate)
  vector = Array.new(num_bits){rand()}
  best = nil
```

```
  max_iterations.times do |iter|
    current = nil
    num_samples.times do
      candidate = generate_candidate(vector)
      candidate[:cost] = onemax(candidate[:bitstring])
      current = candidate if current.nil? or candidate[:cost]>current[:cost]
      best = candidate if best.nil? or candidate[:cost]>best[:cost]
    end
    update_vector(vector, current, learn_rate)
    mutate_vector(vector, current, p_mutate, mutate_factor)
    puts " >iteration=#{iter}, f=#{best[:cost]}, s=#{best[:bitstring]}"
  end
  return best
end

if __FILE__ == $0
  num_bits = 64
  max_iterations = 100
  num_samples = 100
  p_mutate = 1.0/num_bits
  mutate_factor = 0.05
  learn_rate = 0.1

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

Listing 1: Population-Based Incremental Learning algorithm in the Ruby Programming Language

9 References

9.1 Primary Sources

The Population-Based Incremental Learning algorithm was proposed by Baluja in a technical report that proposed the base algorithm as well as a number of variants inspired by the Learning Vector Quantization algorithm [2].

9.2 Learn More

Baluja and Caruana provide an excellent overview of PBIL and compare it to the standard Genetic Algorithm, released as a technical report [4] and later published [1]. Baluja provides a detailed comparison between the Genetic algorithm and PBIL on a range of problems and scales in another technical report [3]. Greene provided an excellent account on the applicability of PBIL as a practical optimization algorithm [7]. Höhfeld and Rudolph provide the first theoretical analysis of the technique and provide a convergence proof [8].

10 Conclusions

This report described the Population-Based Incremental Learning algorithm using the standardized 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] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of the International Conference on Machine Learning*, pages 36–46. Morgan Kaufmann, 1995.

[2] 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.

[3] Shumeet Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, School of Computer Science Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, September 1995.

[4] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. Technical Report CMU-CS-95-141, School of Computer Science Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, May 1995.

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

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

[7] J. R. Greene. Population-based incremental learning as a simple versatile tool for engineering optimization. In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications*, pages 258–269, 1996.

[8] M. Höhfeld and G. Rudolph. Towards a theory of population based incremental learning. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 1–5. IEEE Press, 1997.