Cultural Algorithm[*]

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

November 21, 2010
Technical Report: CA-TR-20101121a-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 Cultural Algorithm using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Cultural. 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 Cultural Algorithm using the standardized algorithm description template.

2 Name

Cultural Algorithm, CA

3 Taxonomy

The Cultural Algorithm is an extension to the field of Evolutionary Computation and may be considered a Meta-Evolutionary Algorithm. It more broadly belongs to the field of Computational Intelligence and Metaheuristics. It is related to other high-order extensions of Evolutionary Computation such as the Memetic Algorithm.

4 Inspiration

The Cultural Algorithm is inspired by the principle of cultural evolution. Culture includes the habits, knowledge, beliefs, customs, and morals of a member of society. Culture does not exist independent of the environment, and can interact with the environment via positive or negative feedback cycles. the study of the interaction of culture in the environment is referred to as Cultural Ecology.

5 Metaphor

The Cultural Algorithm may be explained in the context of the inspiring system. As the evolutionary process unfolds, individuals accumulate information about the world which is communicated to other individuals in the population. Collectively this corpus of information is a knowledge base that members of the population may tap-into and exploit. Positive feedback mechanisms can occur where cultural knowledge indicates useful areas of the environment, information which is passed down between generations, exploited, refined, and adapted as situations change. Additionally, areas of potential hazard may also be communicated through the cultural knowledge base.

6 Strategy

The information processing objective of the algorithm is to improve the learning or convergence of an embedded search technique (typically an evolutionary algorithm) using a higher-order cultural evolution. The algorithm operates at two levels: population level and a cultural level. The population level is like an evolutionary search, where individuals represent candidate solutions, are mostly distinct and their characteristics are translated into an objective or cost function in the problem domain. The second level is the knowledge or believe space where information acquired by generations is stored, and which is accessible to the current generation. A communication protocol is used to allow the two spaces to interact and the types of of information that can be exchanged.

7 Procedure

The focus of the algorithm is the `KnowledgeBase` data structure that records different knowledge types based on the nature of the problem. For example, the structure may be used to record the best candidate solution found as well as generalized information about areas of the search space that are expected to payoff (result in good candidate solutions). This cultural knowledge is discovered by the population-based evolutionary search, and is in turn used to influence subsequent generations. The accept functions constrain the communication of knowledge from the population to the knowledge base.

Algorithm 1 provides a pseudo-code listing of the Cultural Algorithm. The algorithm is abstract, providing flexibility in the interpretation of the processes such as the acceptance of information, the structure of the knowledge base, and the specific embedded evolutionary algorithm.

8 Heuristics

- The Cultural Algorithm was initially used as a simulation tool to investigate Cultural Ecology. It has been adapted for use as an optimization algorithm for a wide variety of

Algorithm 1: Pseudo Code for the Cultural Algorithm.

```
Input: $Problem_{size}$, $Population_{num}$
Output: KnowledgeBase
1  Population ← InitializePopulation($Problem_{size}$, $Population_{num}$);
2  KnowledgeBase ← InitializeKnowledgebase($Problem_{size}$, $Population_{num}$);
3  while ¬StopCondition() do
4      Evaluate(Population);
5      $SituationalKnowledge_{candidate}$ ← AcceptSituationalKnowledge(Population);
6      UpdateSituationalKnowledge(KnowledgeBase, $SituationalKnowledge_{candidate}$);
7      Children ← ReproduceWithInfluence(Population, KnowledgeBase);
8      Population ← Select(Children, Population);
9      $NormativeKnowledge_{candidate}$ ← AcceptNormativeKnowledge(Population);
10     UpdateNormativeKnowledge(KnowledgeBase, $NormativeKnowledge_{candidate}$);
11 end
12 return KnowledgeBase;
```

domains not-limited to constraint optimization, combinatorial optimization, and continuous function optimization.

- The knowledge base structure provides a mechanism for incorporating problem-specific information into the execution of an evolutionary search.
- The acceptance functions that control the flow of information into the knowledge base are typically greedy, only including the best information from the current generation, and not replacing existing knowledge unless it is an improvement.
- Acceptance functions are traditionally deterministic, although probabilistic and fuzzy acceptance functions have been investigated.

9 Code Listing

Listing 1 provides an example of the Cultural 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 $n = 2$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$.

The Cultural Algorithm was implemented based on the description of the Cultural Algorithm Evolutionary Program (CAEP) presented by Reynolds [6]. A real-valued Genetic Algorithm was used as the embedded evolutionary algorithm. The overall best solution is taken as the 'situational' cultural knowledge, whereas the bounds of the top 20% of the best solutions each generation are taken as the 'normative' cultural knowledge. The situational knowledge is returned as the result of the search, whereas the normative knowledge is used to influence the evolutionary process. Specifically, vector the bounds in the normative knowledge are used to define a subspace from which new candidate solutions are uniformly sampled during the reproduction step of the evolutionary algorithm's variation mechanism. A real-valued representation and a binary tournament selection strategy are used by the evolutionary algorithm.

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

def create_random_solution(problem_size, search_space)
  vector = Array.new(problem_size) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
  return {:vector=>vector}
end

def mutate_with_influence(candidate, belief_space, search_space)
  vector = Array.new(candidate[:vector].size)
  candidate[:vector].each_with_index do |c,i|
    range = (belief_space[:normative][i][1] - belief_space[:normative][i][0])
    v = belief_space[:normative][i][0] + rand() * range
    v = search_space[i][0] if v < search_space[i][0]
    v = search_space[i][1] if v > search_space[i][1]
    vector[i] = v
  end
  return {:vector=>vector}
end

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

def initialize_beliefspace(problem_size, search_space)
  belief_space = {}
  belief_space[:situational] = nil
  belief_space[:normative] = Array.new(problem_size) {|i| Array.new(search_space[i])}
  return belief_space
end

def update_beliefspace_situational!(belief_space, best)
  curr_best = belief_space[:situational]
  if curr_best.nil? or best[:fitness] < curr_best[:fitness]
    belief_space[:situational] = best
  end
end

def update_beliefspace_normative!(belief_space, acccepted)
  belief_space[:normative].each_with_index do |bounds,i|
    bounds[0] = acccepted.min{|x,y| x[:vector][i]<=>y[:vector][i]}[:vector][i]
    bounds[1] = acccepted.max{|x,y| x[:vector][i]<=>y[:vector][i]}[:vector][i]
  end
end

def search(max_gens, problem_size, search_space, pop_size, num_accepted)
  # initialize
  pop = Array.new(pop_size) { create_random_solution(problem_size, search_space) }
  belief_space = initialize_beliefspace(problem_size, search_space)
  # evaluate
  pop.each{|c| c[:fitness] = objective_function(c[:vector])}
  best = pop.sort{|x,y| x[:fitness] <=> y[:fitness]}.first
  # update situational knowledge
  update_beliefspace_situational!(belief_space, best)
  max_gens.times do |gen|
    # create next generation
    children = Array.new(pop_size) {|i| mutate_with_influence(pop[i], belief_space,
        search_space) }
    # evaluate
    children.each{|c| c[:fitness] = objective_function(c[:vector])}
    best = children.sort{|x,y| x[:fitness] <=> y[:fitness]}.first
    # update situational knowledge
    update_beliefspace_situational!(belief_space, best)
    # select next generation
    pop = Array.new(pop_size) { binary_tournament(children + pop) }
    # update normative knowledge
    pop.sort!{|x,y| x[:fitness] <=> y[:fitness]}
    acccepted = pop[0...num_accepted]
    update_beliefspace_normative!(belief_space, acccepted)
    # user feedback
    puts " > generation=#{gen}, f=#{belief_space[:situational][:fitness]}"
  end
  return belief_space[:situational]
end

if __FILE__ == $0
  problem_size = 2
  search_space = Array.new(problem_size) {|i| [-5, +5]}
  max_generations = 200
  population_size = 100
  num_accepted = (population_size*0.20).round

  best = search(max_generations, problem_size, search_space, population_size, num_accepted)
  puts "done! Solution: f=#{best[:fitness]}, s=#{best[:vector].inspect}"
end
```

Listing 1: Cultural Algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

The Cultural Algorithm was proposed by Reynolds in 1994 that combined the method with the Version Space Algorithm (a binary string based Genetic Algorithm), where generalizations of individual solutions were communicated as cultural knowledge in the form of schema patterns (strings of 1's, 0's and #'s, where '#' represents either a 1 or a 0) [5].

10.2 Learn More

Chung and Reynolds provide a study of the Cultural Algorithm on a testbed of constraint satisfaction problems [3]. Reynolds provides a detailed overview of the history of the technique as a book chapter that presents the state of the art and summaries of application areas including concept learning and continuous function optimization [6]. Coello Coello and Becerra proposed a variation of the Cultural Algorithm that uses Evolutionary Programming as the embedded weak search method, for use with Multi-Objective Optimization problems [4].

11 Conclusions

This report described the Cultural Algorithm using the standardized algorithm description 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] Chan-Jin Chung and Robert G. Reynolds. A testbed for solving optimization problems using cultural algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 225–236, 1996.

[4] C. A. Coello Coello and R. L. Becerra. Evolutionary multiobjective optimization using a cultural algorithm. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 6–13. IEEE Press, 2003.

[5] R. G. Reynolds. An introduction to cultural algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 131–139. World Scienfific Publishing, 1994.

[6] Robert G. Reynolds. *New ideas in optimization*, chapter Cultural algorithms: theory and applications, pages 367–378. McGraw-Hill Ltd., 1999.