Artificial Immune Network*

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

November 14, 2010
Technical Report: CA-TR-20101114-1

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 Artificial Immune Network algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Artificial, Immune, Network`

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 Immune Network Algorithm using the standardized template.

2 Name

Artificial Immune Network, aiNet, Optimization Artificial Immune Network, opt-aiNet.

3 Taxonomy

The Artificial Immune Network algorithm (aiNet) is a Immune Network Algorithm from the field of Artificial Immune Systems. It is related to other Artificial Immune System algorithms such as the Clonal Selection Algorithm, the Negative Selection Algorithm, and the Dendritic Algorithm. Artificial Immune Network algorithm includes the base version and the extension for optimization problems called the Optimization Artificial Immune Network algorithm (opt-aiNet).

4 Inspiration

The Artificial Immune Network algorithm is inspired by the Immune Network theory of the acquired immune system. The clonal selection theory of acquired immunity accounts for the adaptive behavior of the immune system including the ongoing selection and proliferation of cells that select-for potentially harmful (and typically foreign) material in the body. A concern of the clonal selection theory is that it presumes that the repertoire of reactive cells remains idle when there are no pathogen to which to respond. Jerne proposed an Immune Network Theory (Idiotypic Networks) where immune cells are not at rest in the absence of pathogen, instead antibody and immune cells recognize and respond to each other [9, 10, 11].

The Immune Network theory proposes that antibody (both free floating and surface bound) possess idiotopes (surface features) to which the receptors of other antibody can bind. As a result of receptor interactions, the repertoire becomes dynamic, where receptors continually both inhibit and excite each other in complex regulatory networks (chains of receptors). The theory suggests that the clonal selection process may be triggered by the idiotopes of other immune cells and molecules in addition to the surface characteristics of pathogen, and that the maturation process applies both to the receptors themselves the idiotopes which they expose.

5 Metaphor

The immune network theory has interesting resource maintenance and signaling information processing properties. The classical clonal selection and negative selection paradigms integrate the accumulative and filtered learning of the acquired immune system, whereas the immune network theory proposes an additional order of complexity between the cells and molecules under selection. In addition to cells that interact directly with pathogen, there are cells that interact with those reactive cells and with pathogen indirectly, in successive layers such that networks of activity for higher-order structures such as internal images of pathogen (promotion), and regulatory networks (so-called anti-idiotopes and anti-anti-idiotopes).

6 Strategy

The objective of the immune network process is to prepare a repertoire of discrete pattern detectors for a given problem domain, where better performing cells suppress low-affinity (similar) cells in the network. This principle is achieved through an interactive process of exposing the population to external information to which it responds with both a clonal selection response and internal meta-dynamics of intra-population responses that stabilizes the responses of the population to the external stimuli.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the Optimization Artificial Immune Network algorithm (opt-aiNet) for minimizing a cost function.

8 Heuristics

- aiNet is designed for unsupervised clustering, where as the opt-aiNet extension was designed for pattern recognition and optimization, specifically multi-modal function optimization.
- The amount of mutation of clones is proportionate to the affinity of the parent cell with the cost function (better fitness, lower mutation).

Algorithm 1: Pseudo Code for the Optimization Artificial Immune Network algorithm (opt-aiNet).

```
Input: Population_size, ProblemSize, N_clones, N_random, AffinityThreshold
Output: S_best
Population ← InitializePopulation(Population_size, ProblemSize);
while ¬StopCondition() do
    EvaluatePopulation(Population);
    S_best ← GetBestSolution(Population);
    Progeny ← 0;
    Cost_avg ← CalculateAveragePopulationCost(Population);
    repeat
        clone  foreach Cell_i ∈ Population do
            Clones ← CreateClones(Cell_i, N_clones);
            foreach Clone_i ∈ Clones do
                Clone_i ← MutateRelativeToFitnessOfParent(Clone_i, Cell_i);
            end
            EvaluatePopulation(Clones);
            Progeny ← GetBestSolution(Clones);
        end
    until CalculateAveragePopulationCost(Population) ≤ Cost_avg ;
    SupressLowAffinityCells(Progeny, AffinityThreshold);
    Progeny ← CreateRandomCells(N_random);
    Population ← Progeny;
end
return S_best;
```

- The addition of random cells each iteration adds a random-restart like capability to the algorithms.
- Suppression based on cell similarity provides a mechanism for reducing redundancy.
- The population size is dynamic, and if it continues to grow it may be an indication of a problem with many local optima or that the affinity threshold may needs to be increased.
- Affinity proportionate mutation is performed using $c\prime = c + \alpha \times N(1, 0)$ where $\alpha = \frac{1}{\beta} \times exp(-f)$, N is a Guassian random number, and f is the fitness of the parent cell, β controls the decay of the function and can be set to 100.
- The affinity threshold is problem and representation specific, for example a $AffinityThreshold$ may be set to an arbitrary value such as 0.1 on a continuous function domain, or calculated as a percentage of the size of the problem space.
- The number of random cells inserted may be 40% of the population size.
- The number of clones created for a cell may be small, such as 10.

9 Code Listing

Listing 1 provides an example of the Optimization Artificial Immune Network (opt-aiNet) implemented in the Ruby Programming Language. The demonstration problem is an instance of a continuous function optimization that seeks $min f(x)$ 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 algorithm is an implementation based on the specification by de Castro and Von Zuben [4].

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

def random_vector(problem_size, search_space)
  return Array.new(problem_size) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
end

def random_gaussian
  u1 = u2 = w = g1 = g2 = 0
  begin
    u1 = 2 * rand() - 1
    u2 = 2 * rand() - 1
    w = u1 * u1 + u2 * u2
  end while w >= 1
  w = Math.sqrt((-2 * Math.log(w)) / w)
  g2 = u1 * w;
  g1 = u2 * w;
  return g1
end

def clone(parent)
  v = Array.new(parent[:vector].length) {|i| parent[:vector][i]}
  return {:vector=>v}
end

def mutate(beta, child, rank)
  child[:vector].each_with_index do |v, i|
    alpha = (1.0/beta) * Math.exp(-rank)
    child[:vector][i] = v + alpha * random_gaussian
  end
end

def clone_cell(beta, num_clones, parent, rank)
  clones = []
  num_clones.times {clones << clone(parent)}
  clones.each {|clone| mutate(beta, clone, rank)}
  clones.each{|c| c[:cost] = objective_function(c[:vector])}
  clones.sort!{|x,y| x[:cost] <=> y[:cost]}
  return clones.first
end

def average_cost(population)
  sum = 0.0
  population.each do |p|
    sum += p[:cost]
  end
  return sum / population.length.to_f
end

def euclidean_distance(c1, c2)
  sum = 0.0
  c1[:vector].each_with_index do |v, i|
    sum += (v - c2[:vector][i])**2.0
  end
  return Math.sqrt(sum)
end

def get_neighborhood(cell, pop, affinity_thresh)
```

```ruby
  neighbors = []
  pop.each do |p|
    next if p.equal?(cell)
    neighbors << p if euclidean_distance(p, cell) < affinity_thresh
  end
  return neighbors
end

def affinity_supress(population, affinity_thresh)
  pop = []
  population.each do |cell|
    neighbors = get_neighborhood(cell, population, affinity_thresh)
    neighbors.sort!{|x,y| x[:cost] <=> y[:cost]}
    pop << cell if neighbors.empty? or cell.equal?(neighbors.first)
  end
  return pop
end

def search(problem_size, search_space, max_gens, pop_size, num_clones, beta, num_rand, affinity_thresh)
  pop = Array.new(pop_size) {|i| {:vector=>random_vector(problem_size, search_space)} }
  pop.each{|c| c[:cost] = objective_function(c[:vector])}
  gen, best = 0, nil
  max_gens.times do |gen|
    pop.each{|c| c[:cost] = objective_function(c[:vector])}
    pop.sort!{|x,y| x[:cost] <=> y[:cost]}
    best = pop.first if best.nil? or pop.first[:cost] < best[:cost]
    avgCost, progeny = average_cost(pop), nil
    begin
      progeny = []
      pop.each_with_index {|cell, i| progeny << clone_cell(beta, num_clones, cell, i+1)}
    end until average_cost(progeny) < avgCost
    pop = affinity_supress(progeny, affinity_thresh)
    num_rand.times {pop << {:vector=>random_vector(problem_size, search_space)}}
    puts " > gen #{gen+1}, popSize=#{pop.size}, fitness=#{best[:cost]}"
  end
  return best
end

if __FILE__ == $0
  problem_size = 2
  search_space = Array.new(problem_size) {|i| [-5, +5]}
  max_gens = 200
  pop_size = 20
  num_clones = 10
  beta = 100
  num_rand = 1
  affinity_thresh = 0.5

  best = search(problem_size, search_space, max_gens, pop_size, num_clones, beta, num_rand, affinity_thresh)
  puts "done! Solution: f=#{best[:cost]}, s=#{best[:vector].inspect}"
end
```

Listing 1: Optimization Artificial Immune Network (opt-aiNet) in the Ruby Programming Language

10 References

10.1 Primary Sources

Early works, such as Farmer, et al. [8] suggested at the exploitation of the information processing properties of network theory for machine learning. A seminal network theory based algorithm was proposed by Timmis, et al. for clustering problems called the Artificial Immune Network (AIN) [13] that was later extended and renamed the Resource Limited Artificial Immune System [14] and Artificial Immune Network (AINE) [12]. The Artificial Immune Network (aiNet) algorithm was proposed by de Castro and Von Zuben that extended the principles of the Artificial Immune Network (AIN) and the Clonal Selection Algorithm (CLONALG) and was applied to clustering [5]. The aiNet algorithm was later further extended to optimization domains and renamed opt-aiNet [4].

10.2 Learn More

The authors de Castro and Von Zuben provide a detailed presentation of the aiNet algorithm as a book chapter that includes immunological theory, a description of the algorithm, and demonstration application to clustering problem instances [6]. Timmis and Edmonds provide a careful examination of the opt-aiNet algorithm and propose some modifications and augmentations to improve its applicability and performance for multimodal function optimization problem domains [15]. The authors de Franca, Von Zuben, and de Castro proposed an extension to opt-aiNet that provided a number of enhancements and adapted its capability for for dynamic function optimization problems called dopt-aiNet [7].

11 Conclusions

This report described the Artificial Immune Network algorithm from the field of Artificial Immune systems. Elements of this report were drawn from some of the author's previous works including his dissertation [1].

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. *Clonal Selection as an Inspiration for Adaptive and Distributed Information Processing*. PhD thesis, Complex Intelligent Systems Laboratory, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2008.

[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] L. N. de Castro and J. Timmis. An artificial immune network for multimodal function optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, pages 699–704. IEEE Computer Society, 2002.

[5] L. N. de Castro and F. J. Von Zuben. An evolutionary immune network for data clustering. In *Proceedings Sixth Brazilian Symposium on Neural Networks*, pages 84–89. IEEE Computer Society, 2000.

[6] L. N. de Castro and F. J. Von Zuben. *Data Mining: A Heuristic Approach*, chapter Chapter XII: aiNet: An Artificial Immune Network for Data Analysis, pages 231–259. Idea Group Publishing, 2001.

[7] Fabricio Olivetti de Frana, Fernando J. Von Zuben, and Leandro Nunes de Castro. An artificial immune network for multimodal function optimization on dynamic environments. In *Genetic And Evolutionary Computation Conference*, pages 289–296. ACM Press, 2005.

[8] J. D. Farmer, N. H. Packard, and Alan S. Perelson. The immune system, adaptation, and machine learning. *Physica D*, 22:187–204, 1986.

[9] N. K. Jerne. Clonal selection in a lymphocyte network. In *Cellular Selection and Regulation in the Immune Response*, Society of General Physiologists Series, pages 39–48. Raven Press, 1974.

[10] N. K. Jerne. Towards a network theory of the immune system. *Annales d'immunologie (Annals of Immunology), Institut Pasteur (Paris, France), Societe Francaise d'Immunologie*, 125(C):373–389, 1974.

[11] N. K. Jerne. Idiotypic networks and other preconceived ideas. *Immunological Reviews*, 79:5–24, 1984.

[12] T. Knight and J. Timmis. Aine: An immunological approach to data mining. In T. Lin and Xindon Wu, editors, *First IEEE International Conference on Data Mining (ICDM'01)*, pages 297–304. IEEE Computer Society, 2001.

[13] J. Timmis, M. Neal, and J. Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1):143–150, 2000.

[14] J. Timmis and M. J. Neal. A resource limited artificial immune system for data analysis. *Knowledge Based Systems Journal: Special Issue*, 14(3-4):121–130, 2001.

[15] Jon Timmis and Camilla Edmonds. A comment on opt-ainet: An immune network algorithm for optimisation. In *Lecture Notes in Computer Science*, volume 1, pages 308–317. Springer, 2004.