Artificial Immune Recognition System*

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

November 19, 2010
Technical Report: CA-TR-20101119-1

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 Recognition System using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Artificial, Immune, Recognition, System`

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 [3]. 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 [4]. This report describes the Artificial Immune Recognition System using the standardized algorithm description template.

2 Name

Artificial Immune Recognition System, AIRS

3 Taxonomy

The Artificial Immune Recognition System belongs to the field of Artificial Immune Systems, and more broadly to the field of Computational Intelligence. It is was extended early to the canonical version called the Artificial Immune Recognition System 2 (AIRS2) and provides the basis for extensions such as the Parallel Artificial Immune Recognition System [12]. It is related to other Artificial Immune System algorithms such as the Dendritic Cell Algorithm, the Clonal Selection Algorithm, and the Negative Selection Algorithm.

4 Inspiration

The Artificial Immune Recognition System is inspired by the Clonal Selection theory of acquired immunity. The clonal selection theory credited to Burnet was proposed to account for the behavior and capabilities of antibodies in the acquired immune system [5, 6]. Inspired itself by the principles of Darwinian natural selection theory of evolution, the theory proposes that antigens select-for lymphocytes (both B and T-cells). When a lymphocyte is selected and binds to an antigenic determinant, the cell proliferates making many thousands more copies of itself and differentiates into different cell types (plasma and memory cells). Plasma cells have a short lifespan and produce vast quantities of antibody molecules, whereas memory cells live for an extended period in the host anticipating future recognition of the same determinant. The important feature of the theory is that when a cell is selected and proliferates, it is subjected to small copying errors (changes to the genome called somatic hypermutation) that change the shape of the expressed receptors. It also affects the subsequent determinant recognition capabilities of both the antibodies bound to the lymphocytes cells surface, and the antibodies that plasma cells produce.

5 Metaphor

The theory suggests that starting with an initial repertoire of general immune cells, the system is able to change itself (the compositions and densities of cells and their receptors) in response to experience with the environment. Through a blind process of selection and accumulated variation on the large scale of many billions of cells, the acquired immune system is capable of acquiring the necessary information to protect the host organism from the specific pathogenic dangers of the environment. It also suggests that the system must anticipate (guess) at the pathogen to which it will be exposed, and requires exposure to pathogen that may harm the host before it can acquire the necessary information to provide a defense.

6 Strategy

The information processing objective of the technique is to prepare a set of real-valued vectors to classify patterns. The Artificial Immune Recognition System maintains a pool of memory cells that are prepared by exposing the system to a single iteration of the training data. Candidate memory cells are prepared when the memory cells are insufficiently stimulated for a given input pattern. A process of cloning and mutation of cells occurs for the most stimulated memory cell. The clones compete with each other for entry into the memory pool based on stimulation and on the amount of resources each cell is using. This concept of resources comes from prior work on Artificial Immune Networks, where a single cell (an Artificial Recognition Ball or ARB) represents a set of similar cells. Here, a cell's resources are a function of its stimulation to a given input pattern and the number of clones it may create.

7 Procedure

Algorithm 1 provides a high-level pseudo-code for preparing memory cell vectors using the Artificial Immune Recognition System, specifically the canonical AIRS2. An affinity (distance) measure between input patterns must be defined. For real-valued vectors, this is commonly the Euclidean distance:

$$dist(x, c) = \sum_{i=1}^{n} (x_i - c_i)^2 \quad (1)$$

where n is the number of attributes, x is the input vector and c is a given cell vector. The variation of cells during cloning (somatic hypermutation) occurs inversely proportional to the stimulation of a given cell to an input pattern.

Algorithm 1: Pseudo Code for training the Artificial Immune Recognition System (AIRS2).

Input: InputPatterns, $clone_{rate}$, $mutate_{rate}$, $stim_{thresh}$, $resources_{max}$, $affinity_{thresh}$
Output: $Cells_{memory}$

$Cells_{memory} \leftarrow$ `InitializeMemoryPool(InputPatterns)`;
foreach $InputPattern_i \in$ InputPatterns **do**
  `Stimulate(`$Cells_{memory}$, InputPatterns`)`;
  $Cell_{best} \leftarrow$ `GetMostStimulated(`$InputPattern_i$, $Cells_{memory}$`)`;
  if $Cell_{best}^{class} \neq InputPattern_i^{class}$ **then**
    $Cells_{memory} \leftarrow$ `CreateNewMemoryCell(`$InputPattern_i$`)`;
  else
    $Clones_{num} \leftarrow Cell_{best}^{stim} \times clone_{rate} \times mutate_{rate}$;
    $Cells_{clones} \leftarrow Cell_{best}$;
    for i **to** $Clones_{num}$ **do**
      $Cells_{clones} \leftarrow$ `CloneAndMutate(`$Cell_{best}$`)`;
    end
    while `AverageStimulation(`$Cells_{clones}$`)` $\leq stim_{thresh}$ **do**
      foreach $Cell_i \in Cells_{clones}$ **do**
        $Cells_{clones} \leftarrow$ `CloneAndMutate(`$Cell_i$`)`;
      end
      `Stimulate(`$Cells_{clones}$, InputPatterns`)`;
      `ReducePoolToMaximumResources(`$Cells_{clones}$, $resources_{max}$`)`;
    end
    $Cell_c \leftarrow$ `GetMostStimulated(`$InputPattern_i$, $Cells_{clones}$`)`;
    if $Cell_c^{stim} > Cell_{best}^{stim}$ **then**
      $Cells_{memory} \leftarrow Cell_c$;
      if `Affinity(`$Cell_c$, $Cell_{best}$`)` $\leq affinity_{thresh}$ **then**
        `DeleteCell(`$Cell_{best}$, $Cells_{memory}$`)`;
      end
    end
  end
end
return $Cells_{memory}$;

8 Heuristics

- The AIRS was designed as a supervised algorithm for classification problem domains.
- The AIRS is non-parametric, meaning that it does not rely on assumptions about that structure of the function that is is approximating.
- Real-values in input vectors should be normalized such that $x \in [0, 1)$.
- Euclidean distance is commonly used to measure the distance between real-valued vectors (affinity calculation), although other distance measures may be used (such as dot product), and data specific distance measures may be required for non-scalar attributes.

- Cells may be initialized with small random values or more commonly with values from instances in the training set.
- A cell's affinity is typically minimizing, where as a cells stimulation is maximizing and typically $\in [0, 1]$.

9 Code Listing

Listing 1 provides an example of the Artificial Immune Recognition System implemented in the Ruby Programming Language. The problem is a contrived classification problem in a 2-dimensional domain $x \in [0, 1], y \in [0, 1]$ with two classes: 'A' ($x \in [0, 0.4999999], y \in [0, 0.4999999]$) and 'B' ($x \in [0.5, 1], y \in [0.5, 1]$).

The algorithm is an implementation of the AIRS2 algorithm [11]. An initial pool of memory cells is created, one cell for each class. Euclidean distance divided by the maximum possible distance in the domain is taken as the affinity and stimulation is taken as $1.0 - affinity$. The meta-dynamics for memory cells (competition for input patterns) is not performed and may be added into the implementation as an extension.

```
def random_vector(minmax)
  return Array.new(minmax.length) do |i|
    minmax[i][0] + ((minmax[i][1] - minmax[i][0]) * rand())
  end
end

def generate_random_pattern(domain)
  class_label = domain.keys[rand(domain.keys.length)]
  pattern = {:class_label=>class_label}
  pattern[:vector] = random_vector(domain[class_label])
  return pattern
end

def create_cell(vector, class_label)
  return {:class_label=>class_label, :vector=>vector}
end

def initialize_cells(domain)
  memory_cells = []
  domain.keys.each do |key|
    memory_cells << create_cell(random_vector([[0,1],[0,1]]), key)
  end
  return memory_cells
end

def euclidean_distance(v1, v2)
  sum = 0.0
  v1.each_with_index do |v, i|
    sum += (v1[i]-v2[i])**2.0
  end
  return Math.sqrt(sum)
end

def stimulate(cells, pattern)
  max_affinity = euclidean_distance([0.0,0.0], [1.0,1.0])
  cells.each do |cell|
    cell[:affinity] = euclidean_distance(cell[:vector], pattern[:vector]) / max_affinity
    cell[:stimulation] = 1.0 - cell[:affinity]
  end
end

def get_most_stimulated_cell(memory_cells, pattern)
```

```
43    stimulate(memory_cells, pattern)
44    return memory_cells.sort{|x,y| y[:stimulation] <=> x[:stimulation]}.first
45  end
46
47  def mutate_cell(cell, best_match)
48    range = 1.0 - best_match[:stimulation]
49    cell[:vector].each_with_index do |v,i|
50      min = [(v-(range/2.0)), 0.0].min
51      max = [(v+(range/2.0)), 1.0].max
52      cell[:vector][i] = min + (rand() * (max-min))
53    end
54    return cell
55  end
56
57  def create_arb_pool(pattern, best_match, clone_rate, mutate_rate)
58    pool = []
59    pool << create_cell(best_match[:vector], best_match[:class_label])
60    num_clones = (best_match[:stimulation] * clone_rate * mutate_rate).round
61    num_clones.times do
62      cell = create_cell(best_match[:vector], best_match[:class_label])
63      mutate_cell(cell, best_match)
64      pool << cell
65    end
66    return pool
67  end
68
69  def competition_for_resournces(pool, clone_rate, max_resources)
70    pool.each {|cell| cell[:resources] = cell[:stimulation] * clone_rate}
71    pool.sort!{|x,y| x[:resources] <=> y[:resources]}
72    total_resources = pool.inject(0.0){|sum,cell| sum + cell[:resources]}
73    while total_resources > max_resources
74      cell = pool.delete_at(pool.length-1)
75      total_resources -= cell[:resources]
76    end
77  end
78
79  def refine_arb_pool(pool, pattern, stim_thresh, clone_rate, max_resources)
80    mean_stim, candidate = 0.0, nil
81    begin
82      stimulate(pool, pattern)
83      candidate = pool.sort{|x,y| y[:stimulation] <=> x[:stimulation]}.first
84      mean_stim = pool.inject(0.0){|sum,cell| sum + cell[:stimulation]} / pool.size.to_f
85      if mean_stim < stim_thresh
86        candidate = competition_for_resournces(pool, clone_rate, max_resources)
87        pool.size.times do |i|
88          cell = create_cell(pool[i][:vector], pool[i][:class_label])
89          mutate_cell(cell, pool[i])
90          pool << cell
91        end
92      end
93    end until mean_stim >= stim_thresh
94    return candidate
95  end
96
97  def add_candidate_to_memory_pool(candidate, best_match, memory_cells)
98    if candidate[:stimulation] > best_match[:stimulation]
99      memory_cells << candidate
100   end
101 end
102
103 def train_system(memory_cells, domain, num_patterns, clone_rate, mutate_rate, stim_thresh,
        max_resources)
104   num_patterns.times do |i|
```

```ruby
    pattern = generate_random_pattern(domain)
    best_match = get_most_stimulated_cell(memory_cells, pattern)
    if best_match[:class_label] != pattern[:class_label]
      memory_cells << create_cell(pattern[:vector], pattern[:class_label])
    elsif best_match[:stimulation] < 1.0
      pool = create_arb_pool(pattern, best_match, clone_rate, mutate_rate)
      candidate = refine_arb_pool(pool, pattern, stim_thresh, clone_rate, max_resources)
      add_candidate_to_memory_pool(candidate, best_match, memory_cells)
    end
    puts " > iteration#{i+1} memory_cells=#{memory_cells.size}"
  end
end

def classify_pattern(memory_cells, pattern)
  stimulate(memory_cells, pattern)
  return memory_cells.sort{|x,y| y[:stimulation] <=> x[:stimulation]}.first
end

def test_system(memory_cells, domain)
  correct = 0
  100.times do
    pattern = generate_random_pattern(domain)
    best = classify_pattern(memory_cells, pattern)
    correct += 1 if best[:class_label] == pattern[:class_label]
  end
  puts "Finished test with a score of #{correct}/#{100} (#{correct}%)"
end

def run(domain, num_patterns, clone_rate, mutate_rate, stim_thresh, max_resources)
  memory_cells = initialize_cells(domain)
  train_system(memory_cells, domain, num_patterns, clone_rate, mutate_rate, stim_thresh,
      max_resources)
  test_system(memory_cells, domain)
end

if __FILE__ == $0
  domain = {"A"=>[[0,0.4999999],[0,0.4999999]],"B"=>[[0.5,1],[0.5,1]]}
  num_patterns = 100
  clone_rate = 10
  mutate_rate = 2.0
  stim_thresh = 0.9
  max_resources = 150

  run(domain, num_patterns, clone_rate, mutate_rate, stim_thresh, max_resources)
end
```

Listing 1: Artificial Immune Recognition System in the Ruby Programming Language

10 References

10.1 Primary Sources

The Artificial Immune Recognition System was proposed in the Masters work by Watkins [14], and later published [15]. Early works included the application the AIRS by Watkins and Boggess to a suite of benchmark classification problems [10], and a similar study by Goodman and Boggess comparing to a conceptually similar approach called Learning Vector Quantization [7].

10.2 Learn More

Marwah and Boggess investigated the algorithm seeking issues that affect the algorithms performance [9]. They compared various variations of the algorithm with modified resource allocation

schemes, tie-handling within the ARB pool, and ARB pool organization. Watkins and Timmis proposed a new version of the algorithm called AIRS2 which became the replacement for AIRS1 [11]. The updates reduced the complexity of the approach while maintaining the accuracy of the results. An investigation by Goodman, et al. into the so called '*source of power*' in AIRS indicated that perhaps the memory cell maintenance procedures played an important role [8]. Watkins, et al. provide a detailed review of the technique and its application [13].

11 Conclusions

This report described the Artificial Immune Recognition System using the standardized algorithm description template. This report was based on prior work by the author investigating the algorithm [1, 2] and the implementation provided for the WEKA machine learning workbench http://wekaclassalgos.sourceforge.net/.

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. Artificial immune recognition system (airs) - a review and analysis. Technical Report 1-01, Centre for Intelligent Systems and Complex Processes (CISCP), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Jan 2005.

[2] 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.

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

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

[5] F. M. Burnet. A modification of jernes theory of antibody production using the concept of clonal selection. *Australian Journal of Science*, 20:67–69, 1957.

[6] F. M. Burnet. *The clonal selection theory of acquired immunity*. Vanderbilt University Press, 1959.

[7] D. Goodman, L. Boggess, and A. Watkins. Artificial immune system classification of multiple-class problems. In A. Buczak, J. Ghosh, M. Embrechts, O. Ersoy, and S Kercel, editors, *Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life*, volume 12, pages 179–184, New York, 2002. ASME Press.

[8] D. Goodman, L. Boggess, and A. Watkins. An investigation into the source of power for airs, an artificial immune classification system. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'03)*, pages 1678–1683, Portland, Oregon, USA, 2003.

[9] G. Marwah and L. Boggess. Artificial immune systems for classification: Some issues. In *First International Conference on Artificial Immune Systems*, pages 149–153–, 2002.

[10] A. Watkins and L. Boggess. A new classifier based on resource limited artificial immune systems. In *Part of the 2002 IEEE World Congress on Computational Intelligence*, pages 1546–1551. IEEE, May 2002.

[11] A. Watkins and J. Timmis. Artificial immune recognition system (airs): Revisions and refinements. In P.J. Bentley, editor, *1st International Conference on Artificial Immune Systems (ICARIS2002)*, pages 173–181. University of Kent at Canterbury Printing Unit, UK, 2002.

[12] A. Watkins and J. Timmis. Exploiting parallelism inherent in airs, an artificial immune classifier. In V. Cutello, P. Bentley, and J. Timmis, editors, *Lecture Notes in Computer Science (LNCS)*, volume 3239, pages 427–438. Springer-Verlag GmbH, 2004.

[13] A. Watkins, J. Timmis, and L. Boggess. Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, September 2004.

[14] A. B. Watkins. Airs: A resource limited artificial immune classifier. Master's thesis, Mississippi State University, USA, 2001.

[15] A. B. Watkins and L. C. Boggess. A resource limited artificial immune classifier. In *Part of the 2002 IEEE World Congress on Computational Intelligence held in Honolulu*, pages 926–931, USA, May 2002. IEEE Computer Society.