Dendritic Cell Algorithm*

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

November 20, 2010
Technical Report: CA-TR-20101120-1

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 Dendritic Algorithm using the standard algorithms template.

Keywords: `Clever, Algorithms, Description, Optimization, Dendritic, Cell, 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 Dendritic Cell Algorithm using the standard algorithms template.

2 Name

Dendritic Cell Algorithm, DCA

3 Taxonomy

The Dendritic Cell Algorithm belongs to the field of Artificial Immune Systems, and more broadly to the field of Computational Intelligence. The Dendritic Cell Algorithm is the basis for extensions such as the Deterministic Dendritic Cell Algorithm (dDCA) [4]. It is generally related to other Artificial Immune System algorithms such as the Clonal Selection Algorithm, and the Immune Network Algorithm.

*© Copyright 2010 Jason Brownlee. Some Rights Reserved. This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 2.5 Australia License.

4 Inspiration

The Dendritic Cell Algorithm is inspired by the Danger Theory of the mammalian immune system, and specifically the role and function of dendritic cells. The Danger Theory was proposed by Matzinger and suggests that the roles of the acquired immune system is to respond to signals of danger, rather than discriminating self from non-self [9, 10]. The theory suggests that antigen presenting cells (such as helper T-cells) activate an alarm signal providing the necessarily co-stimulation of antigen-specific cells to respond. Dendritic cells are a type of cell from the innate immune system that respond to some specific forms of danger signals. There are three main types of dendritic cells: 'immature' that collect parts of the antigen and the signals, 'semi-mature' that are immature cells that internally decide that the local signals represent safe and present the antigen to T-cells resulting in tolerance, and 'mature' cells that internally decide that the local signals represent danger and present the antigen to T-cells resulting in a reactive response.

5 Strategy

The information processing objective of the algorithm is to prepare a set of mature dendritic cells (prototypes) that provide context specific information about how to classify normal and anomalous input patterns. This is achieved as a system of three asynchronous processes of 1) migrating sufficiently stimulated immature cells, 2) promoting migrated cells to semi-mature (safe) or mature (danger) status depending in their accumulated response, and 3) labeling observed patterns as safe or dangerous based on the composition of the sub-population of cells that respond to each pattern.

6 Procedure

Algorithm 1 provides pseudo-code for training a pool of cells in the Dendritic Cell Algorithm, specifically the Deterministic Dendritic Cell Algorithm. Mature migrated cells associate their collected input patterns with anomalies, whereas semi-mature migrated cells associate their collected input patterns as normal. The resulting migrated cells can then be used to classify input patterns as normal or anomalous. This can be done through sampling the cells and using a voting mechanism, or more elaborate methods such as a 'mature context antigen value' (MCAV) which is $\frac{M}{Ag}$ (where M is the number of mature cells with the antigen and Ag is the sum of the exposures to the antigen by those mature cells), which gives a probability of a pattern being an anomaly.

7 Heuristics

- The Dendritic Cell Algorithm is not specifically a classification algorithm, it may be considered a data filtering method for use in anomaly detection problems.
- The canonical algorithm is designed to operate on a single discrete, categorical or ordinal input and two probabilistic specific signals indicating the heuristic danger or safety of the input.
- The `danger` and `safe` signals are problem specific signals of the risk that the input pattern is an anomaly or is normal, both typically $\in [0, 100]$.
- The `danger` and `safe` signals do not have to be reciprocal, meaning they may provide conflicting information.

Algorithm 1: Pseudo Code for the Dendritic Cell Algorithm.

```
Input: InputPatterns, $iterations_{max}$, $cells_{num}$, $MigrationThresh_{bounds}$
Output: MigratedCells
1  ImmatureCells ← InitializeCells($cells_{num}$, $MigrationThresh_{bounds}$);
2  MigratedCells ← 0;
3  for $i = 1$ to $iterations_{max}$ do
4      $P_i$ ← SelectInputPattern(InputPatterns);
5      $k_i$ ← $(Pi_{danger} - 2 \times Pi_{safe})$;
6      $cms_i$ ← $(Pi_{danger} + Pi_{safe})$;
7      foreach $Cell_i \in$ ImmatureCells do
8          UpdateCellOutputSignals($Cell_i$, $k_i$, $cms_i$);
9          StoreAntigen($Cell_i$, $Pi_{antigen}$);
10         if $Celli_{lifespan} \leq 0$ then
11             ReInitializeCell($Cell_i$);
12         else if $Celli_{csm} \geq Celli_{thresh}$ then
13             RemoveCell(ImmatureCells, $Cell_i$);
14             ImmatureCells ← CreateNewCell($MigrationThresh_{bounds}$);
15             if $Celli_k < 0$ then
16                 $Celli_{type}$ ← Mature;
17             else
18                 $Celli_{type}$ ← Semimature;
19             end
20             MigratedCells ← $Cell_i$;
21         end
22     end
23 end
24 return MigratedCells;
```

- The system was designed be used in real-time anomaly detection problems, not just static problem.
- Each cells migration threshold is set separately, typically $\in [5, 15]$

8 Code Listing

Listing 1 provides an example of the Dendritic Cell Algorithm implemented in the Ruby Programming Language, specifically the Deterministic Dendritic Cell Algorithm (dDCA). The problem is a contrived anomaly-detection problem with ordinal inputs $\in [0, 50]$, where values that divide by 10 with no remainder are considered anomalies. Probabilistic safe and danger signal functions are provided, suggesting danger signals correctly with $P(danger) = 0.70$, and safe signals correctly with $P(safe) = 0.95$.

The algorithm is an implementation of the Deterministic Dendritic Cell Algorithm (dDCA) as described in [11, 4], with verification from [7]. The algorithm was designed to be executed as three asynchronous processes in a real-time or semi-real time environment. For demonstration purposes, the implementation separated out the three main processes and executed the sequentially as a training and cell promotion phase followed by a test (labeling phase).

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

```ruby

def construct_pattern(class_label, domain, p_safe, p_danger)
  set = domain[class_label]
  selection = rand(set.size)
  pattern = {}
  pattern[:class_label] = class_label
  pattern[:input] = set[selection]
  pattern[:safe] = (rand() * p_safe * 100)
  pattern[:danger] = (rand() * p_danger * 100)
  return pattern
end

def generate_pattern(domain, p_anomaly, p_normal)
  pattern = nil
  if rand() < 0.5
    pattern = construct_pattern("Anomaly", domain, 1.0-p_normal, p_anomaly)
    puts ">Generated Anomoly [#{pattern[:input]}]"
  else
    pattern = construct_pattern("Normal", domain, p_normal, 1.0-p_anomaly)
  end
  return pattern
end

def initialize_cell(thresh, cell={})
  cell[:lifespan] = 100.0
  cell[:k] = 0.0
  cell[:cms] = 0.0
  cell[:migration_threshold] = thresh[0] + ((thresh[1]-thresh[0]) * rand())
  cell[:antigen] = {}
  return cell
end

def store_antigen(cell, input)
  if cell[:antigen][input].nil?
    cell[:antigen][input] = 1
  else
    cell[:antigen][input] += 1
  end
end

def expose_cell(cell, cms, k, pattern, threshold)
  cell[:cms] += cms
  cell[:k] += k
  cell[:lifespan] -= cms
  store_antigen(cell, pattern[:input])
  if cell[:lifespan] <= 0
    initialize_cell(threshold, cell)
  end
end

def can_cell_migrate?(cell)
  return (cell[:cms]>=cell[:migration_threshold] and !cell[:antigen].empty?)
end

def expose_all_cells(cells, pattern, threshold)
  migrate = []
  cms = (pattern[:safe] + pattern[:danger])
  k = pattern[:danger] - (pattern[:safe] * 2.0)
  cells.each do |cell|
    expose_cell(cell, cms, k, pattern, threshold)
    if can_cell_migrate?(cell)
      migrate << cell
      cell[:class_label] = (cell[:k]>0) ? "Anomaly" : "Normal"
    end
  end
  return migrate
end

def train_system(domain, max_iter, num_cells, p_anomaly, p_normal, threshold)
  immature_cells = Array.new(num_cells){ initialize_cell(threshold) }
  migrated = []
  max_iter.times do |iter|
    pattern = generate_pattern(domain, p_anomaly, p_normal)
    migrants = expose_all_cells(immature_cells, pattern, threshold)
    migrants.each do |cell|
      immature_cells.delete(cell)
      immature_cells << initialize_cell(threshold)
      migrated << cell
    end
    puts "> iter=#{iter} new=#{migrants.size}, migrated=#{migrated.size}"
  end
  return migrated
end

def classify_pattern(migrated, pattern, response_size)
  input = pattern[:input]
  num_cells, num_antigen = 0, 0
  migrated.each do |cell|
    if cell[:class_label] == "Anomoly" and !cell[:antigen][input].nil?
      num_cells += 1
      num_antigen += cell[:antigen][input]
    end
  end
  mcav = num_cells.to_f / num_antigen.to_f
  return (mcav>0.5) ? "Anomaly" : "Normal"
end

def test_system(migrated, domain, p_anomaly, p_normal, response_size)
  correct = 0
  100.times do
    pattern = construct_pattern("Normal", domain, p_normal, 1.0-p_anomaly)
    class_label = classify_pattern(migrated, pattern, response_size)
    correct += 1 if class_label == "Normal"
  end
  puts "Finished testing Normal inputs #{correct}/#{100} (#{correct}%)"
  correct = 0
  100.times do
    pattern = construct_pattern("Anomaly", domain, 1.0-p_normal, p_anomaly)
    class_label = classify_pattern(migrated, pattern, response_size)
    correct += 1 if class_label == "Anomaly"
  end
  puts "Finished testing Anomaly inputs #{correct}/#{100} (#{correct}%)"
end

def run(domain, max_iter, num_cells, p_anomaly, p_normal, threshold, response_size)
  migrated = train_system(domain, max_iter, num_cells, p_anomaly, p_normal, threshold)
  test_system(migrated, domain, p_anomaly, p_normal, response_size)
end

if __FILE__ == $0
  domain = {}
  domain["Normal"] = Array.new(50){|i| i}
  domain["Anomaly"] = Array.new(5){|i| (i+1)*10}
  domain["Normal"] = domain["Normal"] - domain["Anomaly"]
  p_anomaly = 0.70
  p_normal = 0.95
  iterations = 100
  num_cells = 20
  threshold = [5,15]
  response_size = 10

  run(domain, iterations, num_cells, p_anomaly, p_normal, threshold, response_size)
end
```

Listing 1: Deterministic Dendritic Cell Algorithm (dDCA) in the Ruby Programming Language

9 References

9.1 Primary Sources

The Dendritic Cell Algorithm was proposed by Greensmith, Aickelin and Cayzer describing the inspiring biological system and providing experimental results on a classification problem [6]. This work was followed shortly by a second study into the algorithm by Greensmith, Twycross, and Aickelin, focusing on computer security instances of anomaly detection and classification problems [8].

9.2 Learn More

The Dendritic Cell Algorithm was the focus of Greensmith's thesis, which provides a detailed discussion of the methods abstraction from the inspiring biological system, and a review of the technique's limitations [3]. A formal presentation of the algorithm is provided by Greenwmith et al. [7]. Greensmith and Aickelin proposed the Deterministic Dendritic Cell Algorithm (dDCA) that seeks to remove some of the stochastic decisions from the method, reduce the complexity and to make it more amenable to analysis [4]. Stibor, et al. provide a theoretical analysis of the Deterministic Dendritic Cell Algorithm, considering the discrimination boundaries of single dendrite cells in the system [11]. Greensmith and Aickelin provide a detailed overview of the Dendritic Cell Algorithm focusing in the information processing principles of the inspiring biological systems as a book chapter [5].

10 Conclusions

This report described the Dendritic Cell Algorithm using the standard algorithms template. The description of the Deterministic Dendritic Cell Algorithm in the literature was found to be incomplete and ambiguous. As such, the current implementation of the algorithm in the Ruby programming language, while it executes, is considered incomplete. Similarly, the pseudo code listing may too require amendment. It is suggested that the author of the approach be contacted and as such, remains an area for further work.

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 (somewhat) wrong by default. 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] J. Greensmith. *The Dendritic Cell Algorithm*. PhD thesis, University of Nottingham, 2007.

[4] J. Greensmith and U. Aickelin. The deterministic dendritic cell algorithm. In *Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS 2007)*, pages 291–302, 2008.

[5] J. Greensmith and U. Aickelin. *Human-Centric Information Processing Through Granular Modelling*, chapter Artificial Dendritic Cells: Multi-faceted Perspectives, pages 375–395. Springer, 2009.

[6] J. Greensmith, U. Aickelin, and S. Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In *Proceedings of the Fourth International Conference on Artificial Immune Systems (ICARIS-05)*, pages 153–167, 2005.

[7] Julie Greensmith, Uwe Aickelin, and Jamie Twycross. Articulation and clarification of the dendritic cell algorithm. In *Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS 2006)*, pages 404–417, 2006.

[8] Julie Greensmith, Jamie Twycross, and Uwe Aickelin. Dendritic cells for anomaly detection. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2006)*, pages 664–671, 2006.

[9] P. Matzinger. Tolerance, danger, and the extended family. *Annual Review of Immunology*, 12:991–1045, 1994.

[10] P. Matzinger. The danger model: A renewed sense of self. *Science*, 296(5566):301–305, 2002.

[11] Thomas Stibor, Robert Oates, Graham Kendall, and Jonathan M. Garibaldi. Geometrical insights into the dendritic cell algorithms. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009.