SOFT COMPUTING TECHNIQUES FOR INTRUSION DETECTION

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Memphis

Jonatan Gomez Perdomo
August 2004
Dedication

This work is dedicated to my wife Elizabeth, my son Sebastian, my mother María, and my little brother Dennis.
Acknowledgements

To Dr. Dipankar Dasgupta, for his support and continuous encouragement.

To my Dissertation Committee members, Dr. Olfa Nasraoui, Dr. Robert Kozma, and Dr. Stan Fraklin, for their valuable suggestions and constructive criticism.

To the members of the Intelligent Security Systems Research Lab: Fabio, Prasad, Madhavi, Senhua, and Ravi for all their help and friendship.

To the National University of Colombia (Universidad Nacional de Colombia), for the financial support to pursue this doctorate.
Abstract


The main goals of this research is to define a set of data mining techniques based on soft computing concepts and to define a mechanism for integrating them for solving the problem of intrusion detection. This research investigates three different soft data mining techniques: Genetic Fuzzy Rule Based Classifiers (GFRBC), Fuzzy Self/Non-self Discrimination (FD) based on Artificial Immune Systems (AIS) and Fuzzy clustering based on gravitational concepts. First, a general framework for GFRBC that combines fuzzy logic, evolutionary algorithms and class binarization concepts is developed. Two fuzzy class binarizations are introduced, a generic encoding scheme that includes fuzzy set tuning is proposed, a set of genetic operators for variable length chromosomes is defined, and a fitness function is developed. In order to reduce the number of parameters required by the evolutionary algorithm, a hybrid adaptive evolutionary algorithm (HAEA) is developed. It adapts the operator rates while searching for the solution. Next, a fuzzy self/non-self discrimination technique based on AIS that uses the framework of GFRBC is proposed. In order to evolve multiple fuzzy rule detectors, a notion of distance between fuzzy rules is proposed and a niching technique for HAEA is developed. Then, a clustering technique based on gravitational concepts is developed. Such a technique is robust (deals with noise) and unsupervised (determines the number of clusters automatically). It is extended in such a way that some parameters can be removed, other interaction function can be used, and fuzzy analysis can be performed. Finally, an integration and boosting technique is developed. This technique is based on the concept of coupled map lattices. Since each technique is generating a fuzzy level of normalcy (membership to some class, in general), the proposed techniques are considered sites (cells) in the coupled map lattice. Experiments with machine learning data sets suggest that each technique performs well. Also, the results indicated that the boosted approach performs better than the original techniques and it is more robust to damage.
Contents

1 Introduction .................................................. 1
  1.1 Goal .......................................................... 2
  1.2 Main Contributions ...................................... 3
    1.2.1 Genetic Rule Based Classifiers (GRBC) for Misuse Detection .......... 4
    1.2.2 Artificial Immune Systems (AIS) and Fuzzy Logic for Anomaly Detection .... 5
    1.2.3 A Clustering Technique for Anomaly Detection .......................... 6
    1.2.4 Coupled Map Lattice of Intrusion Detection Techniques ................. 6
  1.3 Dissertation Outline ..................................... 7

2 Background: Soft Computing ............................. 8
  2.1 Fuzzy Logic ................................................ 8
    2.1.1 Fuzzy Sets ........................................... 8
    2.1.2 Fuzzy Logic Expressions ................................ 8
    2.1.3 Fuzzy Rules .......................................... 9
  2.2 Evolutionary Algorithms (EA) .......................... 10
  2.3 Biological Immune Systems (BIS) ....................... 11
    2.3.1 Immune Memory ..................................... 11
    2.3.2 Self/non-self Discrimination ............................ 12
  2.4 Coupled Map Lattices (CML) .............................. 12
  2.5 Classification .............................................. 13
    2.5.1 Multi-class Problems and Class Binarization ......................... 13
      2.5.1.1 Unordered Binarization ................................ 13
      2.5.1.2 Ordered Binarization ................................ 14
      2.5.1.3 Round Robin Binarization ................................ 14
3.3.4 Results ................................................................. 32
  3.3.4.1 Genetic Operators Analysis ................................. 33
  3.3.4.2 Comparing HAEA against other Evolutionary Algorithms and Previous
         Reported Results ................................................. 35
3.4 Experiments Using Real Encoding ................................. 36
  3.4.1 Genetic Operators ............................................... 36
  3.4.2 Test Functions .................................................. 36
  3.4.3 Experimental Settings ........................................... 37
  3.4.4 Analysis of Results ............................................. 37
3.5 Niching with HAEA .................................................. 39
  3.5.1 Deterministic Crowding (DC) ................................. 39
  3.5.2 Implementing DC in the HAEA ................................. 41
  3.5.3 Experimentation .................................................. 41
     3.5.3.1 Experimental Settings ................................. 42
     3.5.3.2 Results Analysis ........................................... 42
3.6 Summary ............................................................ 43

4 Evolution of Fuzzy Rule Classifiers ............................... 45
  4.1 Introduction ....................................................... 45
  4.2 Genetic Rule Based Systems (GRBS) ........................... 46
     4.2.1 Pittsburgh and Michigan Approaches ....................... 46
     4.2.2 Rule Encoding .................................................. 47
        4.2.2.1 Conjunctions of Simple Terms ......................... 47
        4.2.2.2 Fixed Condition Structure ............................ 47
        4.2.2.3 Linear-tree Representation with Precedence of Operators
                  ......................................................... 47
        4.2.2.4 Evolution Scope ......................................... 47
  4.3 A Framework for Genetic Rule Based Systems ................... 48
  4.4 Evolution of Fuzzy Rules for a Two-class Problem ............ 48
     4.4.1 Fuzzy Rule Encoding .......................................... 49
        4.4.1.1 Positive Expression (PE) ............................ 50
        4.4.1.2 Encoding Function .................................... 50
        4.4.1.3 Fixed Fuzzy Space .................................... 51
<table>
<thead>
<tr>
<th>Subsection</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1.4</td>
<td>Fuzzy Set Tuning</td>
<td>51</td>
</tr>
<tr>
<td>4.4.1.5</td>
<td>Extended Positive Normal Forms (XPNF)</td>
<td>52</td>
</tr>
<tr>
<td>4.4.1.6</td>
<td>Priority Trees</td>
<td>53</td>
</tr>
<tr>
<td>4.4.1.7</td>
<td>Heaps (Complete Expression Trees)</td>
<td>53</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Genetic Operators</td>
<td>54</td>
</tr>
<tr>
<td>4.4.2.1</td>
<td>Variable Length Simple Point Crossover (VLSPX)</td>
<td>55</td>
</tr>
<tr>
<td>4.4.2.2</td>
<td>Single Bit Mutation (SBM)</td>
<td>55</td>
</tr>
<tr>
<td>4.4.2.3</td>
<td>Gene Addition (ADD)</td>
<td>55</td>
</tr>
<tr>
<td>4.4.2.4</td>
<td>Gene Deletion (DEL)</td>
<td>56</td>
</tr>
<tr>
<td>4.4.2.5</td>
<td>Gene Insertion (INS)</td>
<td>56</td>
</tr>
<tr>
<td>4.4.2.6</td>
<td>Disruption of PEs</td>
<td>57</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Fitness Function</td>
<td>57</td>
</tr>
<tr>
<td>4.4.3.1</td>
<td>Fuzzy Confusion Matrix</td>
<td>57</td>
</tr>
<tr>
<td>4.4.3.2</td>
<td>Definition</td>
<td>59</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Rule Extraction</td>
<td>59</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Experiments with Machine Learning Two-class Problems</td>
<td>59</td>
</tr>
<tr>
<td>4.4.5.1</td>
<td>Experimental Settings</td>
<td>59</td>
</tr>
<tr>
<td>4.4.5.2</td>
<td>Comparison of Encoding Schemes</td>
<td>60</td>
</tr>
<tr>
<td>4.4.5.3</td>
<td>Comparison of Fuzzy Logic Operators</td>
<td>61</td>
</tr>
<tr>
<td>4.4.5.4</td>
<td>Fuzzy Sets Tuning</td>
<td>61</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Fuzzy Rule Complexity</td>
<td>62</td>
</tr>
<tr>
<td>4.5</td>
<td>Fuzzy Class Binarization</td>
<td>63</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Fuzzy Unordered</td>
<td>63</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Fuzzy Round Robin</td>
<td>63</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Experiments with Multi-Class Problems</td>
<td>64</td>
</tr>
<tr>
<td>4.5.3.1</td>
<td>Experimental Settings</td>
<td>64</td>
</tr>
<tr>
<td>4.5.3.2</td>
<td>Results</td>
<td>64</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison with Results Reported in the Literature</td>
<td>65</td>
</tr>
<tr>
<td>4.7</td>
<td>Experiments with the KDDCup99 Intrusion Detection Data Set</td>
<td>65</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Experimental Setting</td>
<td>65</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Analysis of Results</td>
<td>66</td>
</tr>
<tr>
<td>4.8</td>
<td>Summary</td>
<td>67</td>
</tr>
</tbody>
</table>
5 Fuzzy Self/Non-self Discrimination

5.1 Introduction ........................................... 70

5.2 First Approach to Fuzzy Self/Non-self Discrimination (FD) ......................... 71
  5.2.1 Encoding ............................................ 71
  5.2.2 Fitness Function .................................... 71
  5.2.3 Similarity and Distance Definition ........................................... 72
  5.2.4 Self/Non-self Discrimination ........................................... 72
  5.2.5 Approach Disadvantages ........................................... 72

5.3 Structured Fuzzy Self/Non-self Discrimination (S-FD) ..................................... 73
  5.3.1 Encoding ............................................ 73
  5.3.2 Fitness Function .................................... 74
  5.3.3 Distance Definition ........................................... 74

5.4 Fuzzy Self/Non-self Discrimination with Variable Length Encoding and Fuzzy Sets Tuning (T-FD) ........................................... 74
  5.4.1 Distance Definition ........................................... 75
  5.4.2 Fitness Function .................................... 76
  5.4.3 Additional Considerations ........................................... 76
    5.4.3.1 Maintaining Diversity ........................................... 77
    5.4.3.2 Increasing a Fuzzy Rule Detector Volume ........................................... 77
  5.4.4 Extracting the Final Set of Fuzzy Rule Detectors ........................................... 77

5.5 Experimentation with Machine Learning Data ........................................... 77
  5.5.1 Experimental Settings ........................................... 78
  5.5.2 Results and Analysis ........................................... 78
    5.5.2.1 Comparison between T-FD, S-FD, and FD ........................................... 78
    5.5.2.2 Fuzzy Rule Complexity ........................................... 79
    5.5.2.3 Comparison with Results Reported in the Literature ........................................... 79

5.6 Experimentation with Intrusion Detection Data Sets ........................................... 80
  5.6.1 Darpa 99 ........................................... 80
    5.6.1.1 Experimental Settings ........................................... 80
    5.6.1.2 Results and Analysis ........................................... 81
  5.6.2 KDDCup99 ........................................... 81
    5.6.2.1 Experimental settings ........................................... 81
5.6.2.2 Results and Analysis ............................... 82
5.7 Summary ................................................. 85

6 Data Clustering using Randomized Interactions between Data Points ........................................ 87
6.1 Introduction ............................................ 87
6.2 Randomized Gravitational Clustering .................. 88
   6.2.1 Newton’s Laws of Motion .......................... 88
      6.2.1.1 1-Dimensional Laws of Motion ................. 88
      6.2.1.2 n-Dimensional Laws of Motion for Straight Line Trajectories ............. 89
   6.2.2 Gravitational Law ................................... 90
6.3 Clustering .................................................. 92
   6.3.1 Time Complexity Analysis .......................... 94
   6.3.2 Comparison with Wright’s work ...................... 95
   6.3.3 Experimentation ....................................... 95
      6.3.3.1 Experimental Settings ......................... 96
      6.3.3.2 Results ........................................ 96
      6.3.3.3 Scalability and Dynamics of the G-Algorithm .................. 98
      6.3.3.4 Sensitivity to $\alpha$ ............................ 98
      6.3.3.5 Sensitivity to the Initial Gravitational force ($G$) .................. 99
      6.3.3.6 Sensitivity to the Gravitational Force Decay ($\Delta(G)$) ......... 99
      6.3.3.7 Sensitivity to Merging Distance ($\varepsilon$) ................ 102
   6.4 RAIN: Clustering based on RAndomized INteractions of data points ......................... 102
      6.4.1 Maximum Distance between Closest Points ........... 103
      6.4.2 Moving Functions ................................... 103
      6.4.3 Setting the Initial Interaction Strength ($G$) ................ 104
      6.4.4 RAIN Time Complexity ............................. 105
      6.4.5 Experimentation ..................................... 105
         6.4.5.1 Experimental Settings ......................... 106
         6.4.5.2 Analysis of Results ......................... 106
   6.5 Classification using Fuzzy Cluster Prototypes ........... 109
      6.5.1 Cluster Statistical Information ..................... 109
      6.5.2 Fuzzy Cluster Membership .......................... 109
6.5.3 Prediction .................................................. 111
6.5.4 Reviewing the Cluster Prototypes ....................... 112
6.5.5 Experimentation ........................................... 112
   6.5.5.1 Experimental Settings ............................... 112
   6.5.5.2 Analysis of Results ................................. 112
   6.5.5.3 Comparing RAIN and T-FD ......................... 113
6.6 Experimentation with Intrusion Detection Data Sets ........ 114
   6.6.1 Darpa 99 ................................................. 114
      6.6.1.1 Experimental Settings ........................... 114
      6.6.1.2 Results and Analysis .............................. 114
   6.6.2 KDDCup99 ............................................... 115
      6.6.2.1 Experimental settings ............................ 115
      6.6.2.2 Results and Analysis .............................. 115
6.7 Summary .................................................... 117

7 CORAL: Cell Organized Recognition Algorithms .............. 119
   7.1 Introduction ............................................. 119
   7.2 Fuzzy Class Binarization using Cell Organized Recognition Algorithms (CORAL-FCB) .. 119
      7.2.1 Experimentation ....................................... 122
         7.2.1.1 Experimental Settings ........................... 122
         7.2.1.2 CORAL-FCB Dynamics ............................ 122
         7.2.1.3 Comparing CORAL-FCB against Fuzzy Round Robin and Fuzzy Unordered Binarization Schemes .... 125
         7.2.1.4 Robustness ....................................... 125
   7.3 CORAL-B: Boosting a Two-class Classification Technique .... 127
      7.3.1 Experimentation ....................................... 128
         7.3.1.1 Experimental Settings ........................... 129
         7.3.1.2 Analysis of Results .............................. 129
   7.4 Summary .................................................... 130

8 Conclusions and Future Work ................................. 131
   8.1 Main Contributions ....................................... 131
   8.2 Future Work ................................................ 135
List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Fuzzy Logic Operators</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Confusion Matrix</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Performance Metrics</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Machine Learning Testbed</td>
<td>24</td>
</tr>
<tr>
<td>2.5</td>
<td>Classes in 10% of the KDDCup 99</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Binary function tested</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance reached by HA EA on the binary test functions using different set of operators</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison of HA EA performance on binary functions</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Real functions tested</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>Solutions found by the tested EAs with the real functions</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Solutions found by the tested EAs on real functions</td>
<td>39</td>
</tr>
<tr>
<td>3.7</td>
<td>Performance of DC-H A E</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Performance - ranking of different encoding mechanisms.</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>Performance of different sets of fuzzy logic operators</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Performance of fuzzy set tuning against fixed fuzzy sets</td>
<td>62</td>
</tr>
<tr>
<td>4.4</td>
<td>Performance of fuzzy class binarizations</td>
<td>65</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparative performance of the proposed approach</td>
<td>65</td>
</tr>
<tr>
<td>4.6</td>
<td>Composition of the KDDCup 99 training set.</td>
<td>66</td>
</tr>
<tr>
<td>4.7</td>
<td>Accuracy of EFRID and the winner entry in the KDDCup 99 contest</td>
<td>67</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparative performance of T-FD and S-FD on accuracy (%) and number of fuzzy rule detectors generated</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparative performance of the proposed approach</td>
<td>80</td>
</tr>
</tbody>
</table>
5.3 Comparative detection rate of T-FD and S-FD when false alarm is set to 3%. 81
5.4 Detection Rate of T-FD and S-FD in the KDD Cup 99 problem when the false alarm is set to 5%. 82
5.5 Comparative performance of T-FD and EFR. 84

6.1 Comparison of the proposed approach with Wright’s work 95
6.2 Performance of different clustering algorithm on the synthetic data sets 96
6.3 Cluster information generated after the gravitational clustering algorithm is applied. $S_k$ is the set of points that are assigned to the cluster $k$. 109
6.4 Performance reached by RAIN in two-class machine learning data sets. (a) Using all of the data samples, (b) Using only normal samples. 113
6.5 Performance comparison of RAIN and T-FD in the anomaly classification problem. 114
6.6 Comparative detection rate of T-FD and S-FD when false alarm is set to 3%. 114
6.7 Detection Rate of T-FD and S-FD in the KDD Cup 99 problem when the false alarm is set to 5%. 116
6.8 Comparative performance of RAIN, T-FD and EFR. 117

7.1 Performance of CORAL-FCB, Fuzzy Unordered (FUB) and Fuzzy Round Robin (FRRB). 125
7.2 Performance of CORAL-FCB, Fuzzy Unordered (FUB) and Fuzzy Round Robin (FRRB). 129
# List of Figures

1.1 Dissertation Contributions .................................................. 4  
2.1 Fuzzy Space .............................................................................. 9  
3.1 Encoding of the operator probabilities in the chromosome .............. 29  
3.2 Single bit mutation .................................................................... 31  
3.3 Simple transposition operator .................................................... 31  
3.4 Single bit crossover ................................................................... 31  
3.5 Performance evolution using HAEA on binary functions. (a) Royal Road, (b) Deceptive-3 and (c) Deceptive-4 .......................................................... 33  
3.6 Performance (first row) and rates evolution (second row) using HAEA(MXT) on binary functions. (a) Royal Road, (b) Deceptive-3 and (c) Deceptive-4 ........................................... 34  
3.7 Performance evolution (first row) and Rates evolution (second row) using HAEA(MXT) on real functions. (a) Rosenbrock, (b) Schwefel, (c) Rastrigin ........................................ 38  
3.8 Multimodal Functions ............................................................... 42  
3.9 Population evolution for M4 using DC-HAEA ............................... 44  
4.1 Evolutionary Algorithm ............................................................ 49  
4.2 General encoding scheme .......................................................... 50  
4.3 Categorical Fuzzy Sets .............................................................. 51  
4.4 Tuning of Trapezoidal membership functions. ............................... 52  
4.5 Conjunctive Normal Form (CNF) .............................................. 53  
4.6 Priority Trees ........................................................................... 53  
4.7 Heap Representation. (a) Heap. (b) Level enumeration. .................. 54  
4.8 Linear Representation of heaps. ................................................ 54
Chapter 1

Introduction

Nowadays, cyber-terrorism is a potential threat to organizations and countries that have become more dependent on cyber-space [140]. Securing cyber-space is a challenging task which requires innovative solutions to deal with cyber-terrorism in all its forms and manifestations. One of the manifestations of cyber-terrorism is illegal intrusion into the computer resources of an organization. This illegal access has the objective of extracting, modifying or damaging sensitive information. Detecting this threat and responding accordingly are the main tasks of intrusion detection systems.

There are two main approaches to developing intrusion detection systems: misuse detection and anomaly detection [8]. The misuse detection approach uses patterns (called signatures) to detect the presence of known attacks. A signature can be, for example, a portion of code, a pattern of behavior or a sequence of system calls. The anomaly detection approach builds a model of normal behavior of the system. Any system behavior that does not match this model is reported as an anomaly.

Each approach has advantages and disadvantages. Although a misuse detection approach is effective and efficient in detecting known attacks, it rarely detects new attacks. On the other hand, an anomaly detection approach is very good in detecting unknown attacks; however, it may generate a high number of false alarms because it can report unknown normal behavior as anomalous. An ideal intrusion detection system will combine the advantages of each approach to generate a high detection rate while maintaining a low number of false alarms.

Regardless of the approach used, the intrusion detection problem (IDP) has been formulated to classify system behavior patterns into two categories: normal and abnormal. But, is the IDP well formulated? Everyday around the world, information about computer system vulnerabilities is released, automatic tools that exploit such vulnerabilities are developed, and new types of intrusions or attacks are created. Moreover, com-
puter systems are constantly upgraded, new user accounts are opened while others are removed or disabled. The dynamic behavior of computer systems does not allow a precise definition of normalcy.

In order to tackle this difficulty, different soft computing techniques have been applied to the intrusion detection problem. Soft computing covers computational techniques that provide approximate solutions to approximately formulated problems (as is the case of the IDP) [6]. Well-known soft computing techniques are fuzzy logic, neural networks, genetic algorithms, immune computing, cellular automata and coupled map lattices.

Fuzzy logic has been combined with data mining techniques for solving the IDP [111, 115, 145]. The purpose of introducing fuzzy logic is to deal with the fuzzy boundary between the normal and abnormal classes. Neural networks have been extensively applied to the IDP, mainly for modeling the normal behavior of a computer system (users and programs) [51, 40, 21]. Genetic algorithms have been applied in the extraction of patterns that characterize both normal and abnormal behavior [29, 124, 90]. Finally, some ideas of the biological immune system have been applied to the anomaly detection approach [50, 75].

Although all these techniques have been applied with relative success, each of them has issues that should be considered:

- **Scalability**: Some of these approaches cannot be used with large amount of data samples representing normal (and abnormal) behavior because either their time complexity is not linear respectful the size of the data set, or their time complexity is linear, but with a large linear constant.

- **Sensitivity to Parameters**: Each of these approaches has a multitude of parameters that control its performance. In some cases it is difficult to define a good set of parameters.

- **Robustness**: Many of these approaches have been designed as a black box unit. If such a unit fails, the intrusion detection system using it will fail.

1.1 Goal

The purpose of the present work is to develop a robust and scalable intrusion detection system based on soft computing techniques. Specifically, this research concentrates on:

1. **Defining Genetic Rule Based Classifiers (GRBC) for misuse detection.** This work defines a general framework for extracting useful knowledge from a data set in the form of fuzzy rules using evolutionary algorithms. The proposed approach is then applied to network computer audit logs in order to characterize known attacks and normal behavior. Moreover, this work develops an evolutionary algorithm
that adapts the genetic operator probabilities during the evolution of the solutions in order to reduce the number of parameters required by an evolutionary algorithm and therefore reducing the parameter of the intrusion detection system.

2. **Extending Artificial Immune Systems (AIS) for anomaly detection using fuzzy logic.** This work extends the encoding mechanism proposed by Gonzalez and Dasgupta [77] by using fuzzy rules in order to reduce the time and memory complexity of the original AIS approach. Moreover, this extension is based on the GFBS framework developed in the above point, i.e., an evolutionary algorithm with adaptation of genetic operator probabilities. In this way, this work defines a niching strategy for such evolutionary algorithm.

3. **Developing a clustering technique for anomaly detection.** This work develops a clustering technique that provides a positive characterization of the normal space. This clustering technique is similar to the gravitational clustering algorithm proposed by Wright in [161], but reducing the time complexity from cubic to super-linear (less than quadratic). In this way, the proposed gravitational clustering technique will include two new elements: a randomized system’s dynamic simulation and a simulated annealing process for cooling the system. Also, this work will integrate a fuzzy analysis to the clusters generated by the algorithm in order to provide a soft distinction between normal and abnormal.

4. **Integrating multiple approaches using couple map lattice.** This work develops a mechanism for integrating the above approaches into a coupled map lattice in order to improve the performance and robustness of the intrusion detection systems. Basically, each site (cell) of the coupled map lattice has associated a detection technique. Such technique defines the initial value of the site when a data sample is analyzed – the fuzzy prediction on the data sample (behavior). The couple map lattice dynamics defines the final classification assigned to such a data sample (prediction).

**1.2 Main Contributions**

The main contribution of this work is to develop a set of soft computing techniques for solving some intrusion detection problems and integrate them into a single technique, see figure 1.1.

This integrated technique improves the performance and robustness of each single technique. The following is a list of this dissertation contributions.
1.2.1 Genetic Rule Based Classifiers (GRBC) for Misuse Detection

This work proposes a framework for extracting useful knowledge from a data set in the form of fuzzy rules using evolutionary algorithms [62, 64, 70]. Particularly, this work:

- Develops a general encoding scheme for representing rules as linear string of bits using the concept of binary heaps,
- Explores a generalized form of disjunctive and conjunctive normal forms, and a variation of the binary trees with precedence encoding proposed by Dasgupta and Gonzalez in [34],
- Introduces a set of genetic operators for modifying such rules (gene addition, gene deletion, and variable length crossover),
- Defines a fitness function for evaluating the performance of the individuals (fuzzy rules) in evolution,
- Proposes a mechanism for evolving not only fuzzy rules but fuzzy sets too, and
• Discusses different fuzzy class binarization schemes for integrating the fuzzy rules generated into a fuzzy classifier.

The proposed framework is tested with a number of benchmark data sets including the KDDCup99 Network Intrusion Detection data.

Because an evolutionary algorithm has several parameters (probabilities of genetic operators, population size, number of iterations, etc) and the setting of such parameters is considered a time consuming task, this work introduces a new evolutionary algorithm that adapts the genetic operator probabilities (rates) while evolves the solution of the problem [63, 68, 65]. Each individual encodes its own genetic operator rates (probabilities). In every generation, each individual is modified by only one operator that is selected according to its encoded rates. Such rates are updated according to the performance achieved by the offspring (compared to its parent) and a random learning rate. The proposed approach is augmented with a simple transposition operator and tested on a number of benchmark functions.

1.2.2 Artificial Immune Systems (AIS) and Fuzzy Logic for Anomaly Detection

This work proposes an anomaly detection technique based on fuzzy logic and Artificial Immune Systems [71, 72]. The AIS negative selection mechanism proposed by Gonzalez and Dasgupta [77] is extended in such a way that it evolves fuzzy rule detectors instead of hyper-rectangles. Such fuzzy rules will produce a better characterization of the boundary between normal and abnormal using only normal samples. In particular this work:

• Develops a mechanism for evolving fuzzy signatures to detect some cyber attacks,

• Presents three different genetic representation schemes for evolving efficient fuzzy rule detectors,

• Defines a notion of distance between fuzzy rules and fuzzy sets,

• Introduces a niching mechanism for the adaptive evolutionary algorithm, and

• Describes the Fuzzy Rule Detector mechanism in the framework of GRBC in order to evolve not only the fuzzy rules but also the fuzzy sets.

Experiments are conducted with different machine learning and intrusion data sets publicly available.
1.2.3 A Clustering Technique for Anomaly Detection

This work develops an unsupervised and robust clustering technique based on the Gravitational Law and Newton’s second law of motion [69, 66]. This technique automatically determines the number of clusters in the target data set. Each data record in the source data set is considered as an object in the feature space. The objects are moved using gravitational force and the second law of motion. The proposed technique is robust to noise, can be used to generate a partition of the data set at multiple resolution levels, and can also be used to extract seeds to form a good summary of the data. In particular this work,

- Proposes an anomaly detection technique base on the gravitational clustering algorithm,
- Generalizes the gravitational clustering algorithm by applying not only an interaction function based on the Gravitational Law (a polynomial interaction function) but other interaction functions, like exponential function.
- Combines the gravitational clustering technique with fuzzy cluster analysis by assigning a fuzzy membership function to the generated clusters, and
- Studies the automatic setting of some of the clustering algorithm parameters.

Experiments with synthetic and intrusion detection data sets are conducted to show the performance of the proposed clustering technique.

1.2.4 Coupled Map Lattice of Intrusion Detection Techniques

This work presents a technique based on coupled map lattices for integrating the previous developed techniques into a single intrusion detection system [73, 67, 101]. The system is designed in such a way that is robust, distributed, and has a better performance than each one of the integrated techniques. In particular this work:

- Develops a fuzzy class binarization technique based on the concept of coupled map lattices,
- Introduces a boosting technique using coupled map lattices, and
- Integrates the proposed techniques.
1.3 Dissertation Outline

This document presents the research that I have performed.

Chapter 2 gives an overview of the techniques included in these research work. It introduces concepts of soft computing such as fuzzy logic, evolutionary algorithms, coupled map lattices, etc. Also, it includes a brief introduction to data mining, particularly to classification and clustering. A survey of the soft computing techniques that have been applied to intrusion detection is presented at the end of the chapter.

Chapter 3 introduces the evolutionary algorithm that adapts the genetic operator rates at the same time that evolves the solution of the problem. This algorithm is used in both the evolution of fuzzy rules and the immuno-fuzzy approaches.

Chapter 4 presents the proposed genetic rule based classifier for intrusion detection. This describes the encoding mechanisms used, the genetic operators, fitness function, fuzzy binarization schemes, and the concept of fuzzy confusion matrix developed.

Chapter 5 details the structure of the immuno-fuzzy approaches for intrusion detection and compares them against the previous immuno-crisp approach.

Chapter 6 presents the fuzzy randomized gravitational clustering algorithm and its application to intrusion detection.

Chapter 7 introduces the integrated technique based on coupled map lattices. Such technique integrates the evolution of fuzzy rules approach, the immuno-fuzzy technique, and the fuzzy gravitational clustering algorithm for intrusion detection.

Chapter 8 draws some conclusions and future work.
Chapter 2

Background: Soft Computing

2.1 Fuzzy Logic

In 1965, Lofti Zadeh introduced a new type of set theory, called fuzzy sets, that allows the representation of concepts that are not well defined [163]. Fuzzy sets allow an element to belong to a set with a degree of membership (not only a binary degree). This concept is extended to logic, allowing a sentence to be partially true. Since then, fuzzy logic has been applied successfully in solving several classification problems [19, 64].

2.1.1 Fuzzy Sets

In fuzzy logic [163], fuzzy sets define linguistic notions, and membership functions define the truth-value of such linguistic expressions. The membership degree of an object to a fuzzy set takes values between 0 and 1, where 1 means entirely in the set, 0 means entirely not in the set, and other values mean partially in the set. The degree of membership of an object in a fuzzy set is defined as a function where the universe of discourse is the domain, and the interval [0,1] is the range.

A collection of fuzzy sets, called fuzzy space, defines the fuzzy linguistic values or fuzzy-classes that an object can belong to, see figure 2.1. In this way, fuzzy sets allow an object to belong to different classes at the same time (with certain degree of membership).

2.1.2 Fuzzy Logic Expressions

In this paper, the minimal fuzzy logic expression (atomic expression) is defined in terms of fuzzy set memberships. An atomic expression is defined as follows:
Figure 2.1: Fuzzy Space

$x$ is [not] fuzzyset,

where $x$ is an object, and fuzzyset is a fuzzy set that belongs to the defined fuzzy space for the object $x$. The truth-value (TV) of an atomic expression is the degree value of membership of the object to the fuzzy set. Because TVs are expressed by numbers between 0 and 1, (0 means entirely false, 1 means entirely true, and other values partially true), the fuzzy expression evaluation process is reduced to arithmetic operations. Also, for each classical logic operator (and, or, not), there are different fuzzy logic operators, see table 2.1:

### Table 2.1: Fuzzy Logic Operators

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>TRUTH-VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not $\neg p$</td>
<td>$TV(\neg p) = 1 - TV(p)$</td>
</tr>
<tr>
<td>And $p \land q$</td>
<td>$TV(p \land q) = min{TV(p), TV(q)}$, $TV(p \land q) = TV(p) \ast TV(q)$, $TV(p \land q) = \frac{2(1 + TV(p) \ast TV(q))}{TV(p) + TV(q)}$</td>
</tr>
<tr>
<td>Or $p \lor q$</td>
<td>$TV(p \lor q) = max{TV(p), TV(q)}$, $TV(p \lor q) = min{1, TV(p) + TV(q)}$</td>
</tr>
</tbody>
</table>

#### 2.1.3 Fuzzy Rules

Fuzzy rules can be defined in many different ways [153]. We consider a fuzzy rule as:

**IF condition THEN consequent**

where, condition is a complex fuzzy expression, i.e., a logic expression that uses fuzzy logic operators and atomic fuzzy expressions, and consequent is an atomic expression. The TV of a fuzzy rule is given by the TV of its condition part. Therefore, for a given object $x$, and fuzzy rule $R$:

$$TV(R, x) = TV(condition_R, x)$$
2.2 Evolutionary Algorithms (EA)

An evolutionary algorithm (EA) is an optimization technique based on the principles of natural evolution [88]. Although there are several different versions of EAs, such as Evolutionary Strategies (ES), and Genetic Algorithms (GA), an EA has basically the structure shown in algorithm 1. The major differences between them are the encoding scheme used and the evolution mechanisms used into the method \textsc{GeneratePopulation}.

\begin{algorithm}
\caption{Evolutionary Algorithm}
\texttt{EA}( \lambda, \text{terminationCondition} )
\begin{algorithmic}[1]
\State $t_0 = 0$
\State $P_0 = \text{initPopulation}( \lambda )$
\While{\text{terminationCondition}( t, P_t ) is false }
\State $P_{t+1} = \text{GeneratePopulation}( P_t )$
\State $t = t + 1$
\EndWhile
\end{algorithmic}
\end{algorithm}

Here,

- \( \lambda \) is the number of individuals in the initial population, \( \text{initPopulation} \) is a method that generates the individuals of the initial population,
- \( P_t \) is the population at iteration \( t \),
- \( P_e \) is a subset of individuals of \( P_t \) that is selected according to a global selection strategy,
- \( \text{GeneratePopulation} \) is a process that selects a subset of the population (the parents population), applies a set of predefined genetic operators on it and generates the offspring population, and
- \( \text{TerminationCondition} \) is a predicate that defines when the EA has to end.

A parent selection strategy usually takes into account the individual’s fitness and compares it with the entire population (or subpopulation) for parent selection. Tournament and Elitism are well known selection strategies.

In the tournament strategy, a set of \( n \) individuals (the tournament size), is chosen in a uniform random way from the population. The winner in the competition among those individuals is copied to the temporal population. The probability of winning the competition is proportional to the individual fitness measure.

In the elitism strategy the individuals are sorted by fitness and a proportion of the best individuals (the elite percentage) is kept while the remaining individuals are chosen randomly with a probability proportional to their fitness. In this strategy a proportion of the worst individuals is usually not used [59].
2.3 Biological Immune Systems (BIS)

The Biological Immune System (BIS) is a complex system that has the task of protecting the body from entities that can harm [75, 156]. The BIS is composed by several mechanisms that interact in a way not fully understood. Some of these mechanisms are the immune memory and the self/non-self discrimination. Artificial Immune Systems (AIS) are pattern recognition techniques inspired on such mechanisms [75].

2.3.1 Immune Memory

The immune memory mechanism allows the BIS to remember encounters with antigens (harmful entities), in order to produce a fast response towards some previously seen antigens. Some AIS research works have tried to model this mechanism using the Idiotypic Network Theory (INT) developed by Jerne [94]. The INT suggested that the immune memory of a BIS is defined as a network of B-cells that can regulate the number and type of antibodies.

Farmer et al proposed a computational model of the BIS [46]. Based on a simplification of the INT, the model represents antibodies and antigens as binary sequences and uses a set of equations for defining the system’s dynamic [75].

Varela [156] suggested that idiotypic networks can be considered as cognitive networks and compared them against neural networks. Bersini et al [15] proposed a recruitment mechanism for increasing or decreasing the amount of antibodies required in some antigen elimination tasks.

Timmis [151] proposed the Artificial Immune NEtwork (AINE) as a machine learning algorithm. Training data samples are used as antigens and B-cells are data points that are stimulated by such antigens. Highly stimulated B-cells produce clones that are mutated and added to the network. B-cells receiving a low level of stimulus are removed from the network. The final network of B-cells can be considered as a model of the training data samples.

With the Resource limited Artificial Immune Network (RAIN) approach [152], Timmis addressed the AINE problem of B-cell explosion. In RAIN the concept of Artificial Recognition Ball is introduced. A limited number of ARBs are used for grouping a maximum number of B-cells. Therefore, the number of B-cells is limited too.

Nasraoui et al [128] proposed a fuzzy version of the immune network concept. In the Fuzzy AIS, a fuzzy set is associated with each B-cell. This fuzzy set determines the influence area of the B-cell.
2.3.2 Self/non-self Discrimination

The self/non-self discrimination mechanism allows the BIS to discriminate between cells that belongs to the system (self space), and entities that do not belong to the system (antigens or non-self space). This is done by generating T-cells in the Thymus and testing them against self-cells. T-cells that bind some self-cell (complementarity at the protein level) are destroyed because are recognizing self-cells as antigens. T-cells that do not bind any self-cell are allowed to leave the Thymus to perform the detection task.

Forrest et al [50] proposed the Negative Selection (NS) algorithm based on this process. First, a set of detectors is generated (e.g., randomly); then, these detectors are compared against the self (normal) set; finally, those detectors that match any self element are discarded, and those that do not are kept.

Because the NS uses a binary representation for antigens and antibodies, several binary matching rules have been proposed, each one with advantages and disadvantages [78].

Dasgupta and Gonzalez [33] used a real vector representation instead of using binary strings. Elements of self/non-self space are represented by \( n \)-dimensional real vectors. Detectors correspond to hyper-rectangles in \( \mathbb{R}^n \) and have high level representation as rules. A genetic algorithm with a niching technique is applied for evolving the detectors.

Although some effort has been made in integrating fuzzy logic with AIS for solving the intrusion detection problem [71], the encoding scheme used does not allow generation of simple fuzzy rules or the modification of the shape of fuzzy sets.

2.4 Coupled Map Lattices (CML)

Coupled map lattice (CML) theory grew out of studies on collective movements of coupled oscillators [97, 101] and can be seen as extensions of the concept of cellular automaton [160]. A CML can be defined in any \( d \)-dimensional space \((d = 1, 2, ..)\) but in this paper, we consider only 2-dimensional CML of size \( N \times M \).

A 2-D Coupled Map Lattice is a 2-dimensional lattice of size \( N \times M \) such that each site \( x(i, j) \) evolves according to equation 2.1.

\[
x(i, j)_{t+1} = (1 - \varepsilon) f (x_t (i, j)) + \frac{\varepsilon}{s (N_{x(i,j)})} \sum_{y \in N_{x(i,j)}} f (y_t)
\]

(2.1)

where, \( N_{x(i,j)} \) is a collection of sites which defines the neighborhood of site \( x(i, j) \), \( s (N_{x(i,j)}) \) indicates the number of neighbors, \( f \) is a non-linear function that determines the coupling value of a site, and \( \varepsilon \) is a parameter that determines the coupling level or spatial correlation between sites. By varying the coupling
parameter \( \varepsilon \) and the structure of the neighborhood \( N_{x(i,j)} \), it is possible to simulate systems that interact locally up to systems that interact globally [101].

Several non-linear functions have been used into a CML. In the following discussion we use the well-known logistic function:

\[
f(x) = 1 - \alpha x^2
\]  

(2.2)

where, \( 0 \leq \alpha \leq 2 \) is a suitable parameter that allows one to modify the dynamics of the system and introduce chaotic behavior into the CML. In order to introduce chaos to the CML, \( \alpha \) should be set to a value higher than 1.82.

Coupled map lattices have been used for simulating natural dynamic processes such as fluid dynamics. Moreover, Coupled Map Lattices can be seen as a computational model.

### 2.5 Classification

Classification is a supervised learning technique that takes labeled data samples and generates a model (classifier) that classifies new data samples in different predefined groups or classes [80]. Classification has been extensively studied in machine learning and data mining [80, 123, 89]. According to the number of classes, classification problems can be divided into two-class and multi-class problems. Extensive research has been done in solving two-class problems and converting multi-class problems in a set of two-class problems.

#### 2.5.1 Multi-class Problems and Class Binarization

Because several machine-learning techniques have been designed for solving two-class classification problems, but not for solving multi-class problems, some techniques (class binarization) have been developed for transforming a multi-class problem to several two-class problems [55]. This section presents three basic class binarization strategies.

##### 2.5.1.1 Unordered Binarization

An unordered binarization strategy transforms an \( m \)-class problem into \( m \) two-class problems, where the \( i \)-th classifier that is generated using the samples of class \( i \) as positive samples and samples of the other classes \( (j=1..m, j \neq i) \) as negative samples. A new sample is classified according to algorithm 2. PICK_ONE is a conflict resolution strategy for classifying the sample into only one class.
Algorithm 2 Unordered Classification

CLASSIFY(classifier[1..m], sample)
1. winners = ∅
2. for i = 1 to m do
3. if classifier_i classifies sample as positive then
4. winners = winners ∪ {i}
5. return PICK.ONE(winners)

Algorithm 3 Ordered Classification

CLASSIFY(classifier[1..m−1], sample)
1. i = 1
2. while i ≤ m and classifier_i classifies sample as negative do
3. i = i + 1
4. return i

2.5.1.2 Ordered Binarization

An ordered binarization strategy transforms an m-class problem into m-1 binary problems, where the i-th classifier is generated using the samples of class i, (i = 1..m−1), as positive samples and the samples of classes j > i as negative samples. A new sample is classified according to algorithm 3.

2.5.1.3 Round Robin Binarization

A round robin binarization strategy transforms an m-class problem into \( \frac{m(m-1)}{2} \) two-class problems by defining one classifier classifier_{i,j} for each pair of classes \( \langle i, j \rangle \). Each classifier is generated using only the samples of the two corresponding classes. A new sample is classified according to algorithm 4(majority voting).

Algorithm 4 Round robin classification

CLASSIFY(classifier[1..m−1]|i+1..m], sample)
1. for i = 1 to m do
2. vote[i] = 0
3. for i = 1 to m − 1 do
4. for j = i + 1 to m do
5. if classifier_{i,j} classifies sample as i then
6. vote[i] = vote[i] + 1
7. else
8. vote[j] = vote[j] + 1
9. return MAJORITY_VOTING(vote)
2.5.2 Performance Evaluation

Several performance metrics have been defined for determining the quality of a classifier. Most of them are summarized in a confusion matrix. A confusion matrix contains the classification performance (actual versus predicted class) reached by a classifier system over a set of samples [80]. The confusion matrix for a two-class (Negative and Positive) classifier is shown in table 2.2.

<table>
<thead>
<tr>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>PP</td>
<td>PN</td>
</tr>
<tr>
<td>Negative</td>
<td>NP</td>
<td>NN</td>
</tr>
</tbody>
</table>

From the confusion matrix, several performance metrics can be generated. Table 2.3 shows some of them.

<table>
<thead>
<tr>
<th>TERM</th>
<th>DESCRIPTION</th>
<th>EQUATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (AC)</td>
<td>Proportion of well classified samples</td>
<td>[ AC = \frac{PP + NN}{PP + PN + NP + NN} ]</td>
</tr>
<tr>
<td>True Positive (TP)</td>
<td>Proportion of positive samples well classified</td>
<td>[ TP = \frac{PP}{PP + NP} ]</td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>Proportion of negative samples misclassified</td>
<td>[ FP = \frac{NP}{NP + NN} ]</td>
</tr>
<tr>
<td>True Negative (TN)</td>
<td>Proportion of negative samples well classified</td>
<td>[ TN = \frac{NN}{NP + NN} ]</td>
</tr>
<tr>
<td>False Negative (FN)</td>
<td>Proportion of positive samples misclassified</td>
<td>[ FN = \frac{PN}{PP + PN} ]</td>
</tr>
</tbody>
</table>

2.5.3 Anomaly Classification

An anomaly classification problem (anomaly detection) is a two-class classification problem where only samples of one class (usually the negative class) are given. Basically there are three different approaches for anomaly classification:

1. **Negative Characterization**: The negative samples (normal samples) are used for building a model of the negative space; for example, by generating a set of rules (detectors) that can recognize positive samples (abnormal patterns) [33, 76, 32].

2. **Positive characterization**: The negative samples are used for building a model of the negative space; for example, a set of rules that defines the negative space [106, 19, 87].

3. **Artificial anomalies generation**: The negative samples are used for generating artificial anomaly (positive) samples and a classifier learning technique is used for generating the classifier [45, 77].
2.5.4 Receiver Operating Curve (ROC)

There are two elements that define the cost function of an anomaly classifier system: the false alarm (FA) rate (the system considers abnormal a sample that is normal) and the detection rate (DR) (the system considers an abnormal sample as normal). The Receiver Operating Characteristic (ROC) analysis can be applied to an anomaly classifier to evaluate its performance [133]. In the ROC analysis, for anomaly classifier systems that produce a continuous output with respect to some parameter $a$, the coordinate point $(FP, TP)_a$ is plotted in the coordinate system. Here, TP is the detection rate and FP is the false positive rate (the rate of false alarms). This ROC curve can be used to determine when a classifier is good or bad. If the ROC curve of a classifier $A$ dominates the ROC curve of the classifier $B$, then classifier $A$ is better than classifier $B$ [134].

2.6 Clustering

Clustering is an unsupervised learning technique that takes unlabeled data points (data records) and classifies them into different groups or clusters. This is done in such a way that points assigned to the same cluster have high similarity, while the similarity between points assigned to different clusters is low [80]. Different clustering techniques have been developed [80, 116, 17, 137]. The most important types of clustering techniques are hierarchical agglomerative, partition, robust and unsupervised.

2.6.1 Hierarchical Agglomerative Clustering

A hierarchical clustering technique assigns each data sample to one cluster, merges clusters in an iterative process, and stops when there is only one cluster remaining [80, 165]. Clusters are merged based on some distance notion or similarity measure.

2.6.2 Partition Clustering

A partition clustering technique starts with a selection of $k$ samples from the data set to be clustered. Such samples define the initial set of candidate cluster centers. Next, it assigns samples to such clusters based on some distance notion or similarity measure. Finally, the candidate cluster centers are recomputed. This process is repeated until some stopping criteria is achieved [116, 80, 165]. Usually, this stopping criteria is defined according to some optimization function.
2.6.3 Robust Clustering

Many clustering techniques have relied on the assumption that a data set follows a certain distribution and is free of noise. In fact, if noise is introduced, several techniques based on the least square estimate are spoiled [137]; such is the case with \( k \)-means [116] and fuzzy \( k \)-means [17]. Several approaches have tried to tackle this problem; some of them are based on robust statistics [79, 137], and others are based on modifying the objective of the fuzzy centroid mean to make the parameter estimate more robust with respect to noise [102, 14, 53]. A clustering technique that is able to manage noisy information is called robust.

2.6.4 Unsupervised Clustering

Although clustering techniques are unsupervised learning techniques, many of them require the number of clusters in advance. A clustering technique that does not requires the number of clusters in advance is called unsupervised [95, 54, 129].

2.6.5 Soft Clustering

Several clustering techniques have been developed using soft computing concepts [17, 102, 128]. A review of fuzzy clustering techniques can be found in [11] and [126].

2.6.6 Classification with Clustering Techniques

A clustering technique can be used as a classification one by assigning to each cluster the label of the class with more data samples in the cluster. If two or more class labels can be assigned then a conflict resolution strategy can be applied.

Let \( c_i, i = 1, ..., K \) be the collection of \( K \) clusters generated by the clustering algorithm. Let \( T \) be the number of classes in the classification problem, and let \( n_{i,t} \) the number of training data records that belong to class \( t \) that were assigned to cluster \( i \). This strategy basically consist of two steps:

1. **Building the classification model**: Assign to each cluster \( c_i \) the class with more data records assigned to that cluster. Clearly, only the data records used by the clustering algorithm are used (in our case only 1%).

2. **Prediction**: Given an unknown data record, the record is assigned to the closest cluster. The distance between a data record and a cluster is given by the distance between the data record and the center of the cluster.
**Algorithm 5** Gravitational Clustering

**GRAVITATIONAL( x)**

1. \( n = |x| \) //number of data points
2. for i=1 to n do
3. \( m_i = 1 \) // initialize the objects mass
4. while \( n > 1 \) do
5. for i=1 to n do
6. \( z_i = 0 \)
7. for j=1, j \( \neq \) i to n do
8. \( z_i = z_i + \text{DeltaMove}(x_i, x_j, \text{mass}) \) // movement due to \( x_j \)
9. for i=1 to n do
10. \( x_i = x_i + z_i \)
11. for i=1 to \( n - 1 \) do
12. for j=i+1 to n do
13. if \( \text{dist}^2(x_i, x_j) \leq \varepsilon \) then
14. \( m_i = m_i + m_j \) // add to \( x_i \) the mass of \( x_j \)
15. \( x = x - x_j \) // remove \( x_j \) from the system
16. \( n = n - 1 \) // remove the mass of \( x_j \)
17. \( n = n - 1 \) // reduce the number of points

### 2.6.7 Gravitational Clustering

The gravitational clustering algorithm is a hierarchical clustering algorithm based on concepts of field theory in physics [161, 103]. Each data point is considered as a particle in a space exposed to gravitational fields. A unit mass is associated with each data point. Data points are moved toward the cluster centers due to gravitational fields. To determine the new position of a single particle \( x \), the strength of the gravitational field of any other particle on \( x \) is considered. There is a maximum distance that each particle can move. When two particles (initially data points) are close enough to be merged, one of them is removed from the simulation and the mass of the other is increased (the mass of the particle removed is added). The process is stopped when only one particle remains in the system. A draft of the gravitational clustering algorithm is presented in 5.

One of the major issues with the gravitational clustering is its time complexity. It is possible to prove that the gravitational clustering has \( O(n^3) \) time complexity respect to the size of the data set (\( n \)) [103]. Such high complexity is undesirable when clustering huge data sets.

### 2.7 A Survey on Soft-Computing Techniques for Intrusion Detection

The problem of Intrusion Detection (ID) has been studied extensively in computer security [7, 83, 8, 5] and has received a lot of attention in machine learning and data mining [149, 106, 108, 107, 110, 29]. Lee et al [106] used association rules and learning of frequent episodes for extracting consistent and useful patterns
of programs and user behavior from systems audit data; Li et al [110] mined temporal association rules in
terms of multiple time granularities for determining temporal patterns that govern normal behavior; and Fan
et al [45] used information of the computer system under normal conditions for generating artificial abnormal
behavior and growing a decision tree that discriminate normal from abnormal behavior.

This section presents a brief literature review of soft computing techniques applied to intrusion detection.
The review has been organized based on the soft computing technique employed: Fuzzy Logic, Neural Net-
works, Evolutionary Algorithms and Artificial Immune Systems. Coupled Map Lattices have not yet been
applied to intrusion detection yet.

2.7.1 Fuzzy Logic

Lin [111] used a fuzzy-rough set theory for mining repeating patterns in audit data files. Such patterns reflect
user habits and can be used as signatures of computer users in order to detect abuses and misuses of computer
systems.

Bridges and Vaughn [19, 115] introduced a normalization factor in the fuzzy association rules algorithm
proposed by Kuok et al [104]. The modified algorithm is used for mining fuzzy association rules from audit
data sets. Florez et al [48] used prefix-trees for speeding the fuzzy association rule generation.

Bridges and Vaughn [19, 115] modified the algorithm proposed by Mannila and Toivonen [119] for mining
serial frequency episode rules. The modified algorithm is used for characterizing frequent temporal patterns
in audit data sets.

Dickerson et al [41] developed a Fuzzy Intrusion Recognition Engine (FIRE) for detecting malicious
activity in computer networks in a distributed fashion. FIRE is defined as a collection of autonomous agents.
Each agent produces fuzzy information from input sensors. For each observed feature a fuzzy set is generated
using a fuzzy c-means algorithm. Such information is combined by a fuzzy rule based system for determining
the degree of normalcy. Fuzzy rules are hand-coded (generated by an user with a graphical fuzzy tool).
Experiments are conducted with synthetic data sets in three different scenarios: Port and Host scan, DoS
attack and unauthorized services access.

Siraj et al [145] proposed a Fuzzy Cognitive Map (FCM) for capturing causal knowledge that decision
engines use for alert assessment in computer network environments. Some nodes of the FCM (an FCM is a
signed directed graph) represent events and some others represent suspicious attacks. Links between them
indicate the causal relation between events and possible attacks. No experiments are reported.
Cho [24] used a fuzzy rule reasoning mechanism in order to detect an anomaly. The input of the fuzzy reasoning mechanism are Hidden Markov Model (HMM) evaluation values (from different HMM models). The fuzzy rules are designed according to the set of HMM’s. A centroid defuzzification technique is applied for determining the final classification (abnormal or normal). Experiments are conducted for detecting user-to-root attacks on data collected from graduate students.

Shah et al [141] used Principal Component Analysis (PCA) and fuzzy c-medoids clustering for generating a profile of normal activity in a computer system. PCA is used for reducing the dimensionality of the collected data and fuzzy c-medoids is used for generating the profiles representing normalcy. Experiments are conducted on data collected from UNIX and APACHE web servers.

2.7.2 Neural Networks

Neural Networks have been applied to solve the intrusion detection problem since the early 90s. Fox et al [51] used a Self Organizing Map for learning characteristics of normal activity. These characteristics are used to determine when an intrusion is in progress. In particular, the system detects viruses.

Debar et al [40] developed an intrusion detection technique based on recurrent neural networks. The neural network is trained with sequence of user commands in a SUN3-UNIX machine.

Li [109] used an Adaptive Resonance Theory (ART2) neural network for clustering the normal behavior of users. Such behavior is defined in terms of historical profiles of commands that a user executes. When a user does not follow such profile, he/she is considered an intruder. Experiments are conducted with a set of historical profiles from a computer system.

Barrus and Rowe [13] designed a neural network for measuring and determining alert threshold values. Such alerts are interchanged between a collection of autonomous agents. Such autonomous agents work cooperatively for determining intrusions. The architecture of the network is not described, but it receives eight input values (statistic information for a period of time) and produces one of three possible alertness values. Experiments are performed for detecting coordinated dorknob rattling attack.

Cannady [20] developed a Multi Layer Perceptron (MLP) for identifying misuses in computer systems. The MLP had four fully connected layers, nine inputs and two output nodes. Training and testing data sets are generated using a network monitor tool.

Ryan et al [138] proposed a Back Propagation (BP) neural network for identifying users. The neural network is trained with distribution vectors of 100 commands. Several samples of commands distribution vector per user are given to the neural network in the training process.
Ghosh and Schwartzbard [56] developed a back-propagation neural network for monitoring programs behavior. The neural network is determined by three layers: input, hidden, and output. For each program monitored, a neural network must be defined and trained.

Cannady [21] used a Cerebellar Model Articulation Controller (CMAC) neural network for detecting DoS attacks in a simulated environment. A CMAC neural network is a three layer feed-forward neural network that is capable of on-line learning. This property is used for Cannady to recognize new attacks: starting with a Ping-Flood attack and finishing with a UDP Packet Storm attack.

Lippmann and Cunningham [112] developed a technique that combines keyword selection with neural networks. A feed-forward neural network is used for approximating posterior probabilities of telnet sessions with normal actions and with attacks. Experiments are conducted using the DARPA 98 data set.

Rhodes et al [136] explored the application of Kohonen self-organizing maps for characterizing the normal behavior of a computer network. The input of the Kohonen SOM is provided by a monitor stack that uses protocol analyzers for reducing and discriminating the network traffic. The approach is tested with buffer-overflow attacks.

Lee and Heinbuch [105] proposed a hierarchy of Back Propagation neural networks for detecting intrusions. Neural network in low levels are designed and trained with specific assertions about the network traffic. No network traffic data is used for training the neural networks. Neural networks in top-levels are used to combine the detection provided by low-level neural networks in order to increase the detection accuracy. Experiments are conducted with artificially generated data.

Cho [24] used a Self Organizing Map (SOM) for preprocessing the input of a set of Hidden Markov Models. Such HMMs are “small” models of a computer system behavior. The SOM is used for reducing the dimensionality of the network data. Experiments are conducted for detecting user-to-root attacks on data collected from graduate students.

2.7.3 Evolutionary Algorithms

Crosbie [29] developed a distributed detection system based on concepts of autonomous agents and genetic programming. While the autonomous agents carry the intrusion detection task in a distributed fashion, genetic programming is used for evolving the control programs (detection algorithm) of each agent. Tests are conducted in artificial scenarios.

Mischiatti and Neri [124] used REGAL, a distributed genetic algorithm for evolving rules, on intrusion detection data. The performance of REGAL is compared against RIPPER. Experiments are conducted on the
Information Exploration Shootout (IES) contest. This data set contains logs collected at the gateway between a LAN and Internet.

Me [122] considered the intrusion detection problem as an optimization problem and solved it using a genetic algorithm. The solution of the optimization problem is an hypothesis vector that indicates the attack that produces more risk according to the audit trail file. Because the optimization problem is restricted, the fitness function of an individual is defined using penalization strategies. Experiments are conducted with audit trail files in an AIX environment (attacks are simulated).

Helmer et al [84] used a genetic algorithm for selecting a subset of features that can be used for determining intrusions in a computer system. The feature vectors (used by the GA) describe system calls executed by privileged processes. Experiments are conducted using send-mail data.

Sinclair et al [144] used a genetic algorithm for evolving rules representing simple incoming connection patterns. An individual in the population represents combinations of connection attributes and possible “wild characters”. Connection patterns, marked as normal or abnormal, are used in the evolution process. The GA uses a niching strategy for finding several such good rules instead of a single rule.

Chittur [23] used a genetic algorithm for evolving a decision tree with a randomized coefficient describing the data. The decision tree discriminates between normal and abnormal behavior. Experiments are conducted on the KDDCup 99 data set.

Hossain and Bridges [90] proposed a framework for intrusion detection that combines fuzzy logic with genetic algorithms. First a set of fuzzy association rules extracted from audit data. Then, a genetic algorithm is used for tuning the fuzzy sets’ describing the fuzzy association rules. Finally, the set of tuned fuzzy association rules are considered the profile of normal behavior in the computer system. No experiments are conducted.

Marin et al [120] developed a hybrid technique that combines expert systems, clustering techniques, genetic algorithms, and Linear Vector Quantization (LVQ) for generating user profiles. The dimensionality of the data samples is reduced after they are clustered with a genetic algorithm. The LVQ refines the cluster centers once the dimensionality reduction process is done by the GA. Experiments are conducted on sequences of 5000 UNIX commands for a set of 50 users.

2.7.4 Artificial Immune Systems

Approaches based on artificial immune systems (AIS) have been applied successfully to perform anomaly detection [30, 86, 81, 99]. One major difference with other anomaly detection techniques is that AIS builds
a model of the abnormal instead of the normal. This abnormal model is described in terms of a set of anomaly detectors which are generated by an algorithm called negative selection (NS) [50]. Depending on the representation used, it is possible to see these detectors as signatures of unknown attacks that can be combined with signatures generated by a misuse detection approach to produce an integrated IDS.

Dasgupta and Forrest [31] used the NS algorithm for fault detection. They developed an AIS for detecting tool breakage in milling industries. Also, Tyrrell [155] proposed a complete mapping between immune system components and hardware fault tolerance components in order to use the NS algorithm for hardware fault tolerance.

Kim and Bentley [100] combined the negative selection algorithm, a genetic algorithm, and a niching technique for generating network intrusion detectors. Each individual encodes 35 different pairs of information (genes) that are used for determining the level of deviation of a given data sample. Also, the matching rule between detectors and data samples is defined in terms phenotype distance instead of binary strings distance (as in Forrest et al [50]).

Dasgupta and Gonzalez [33] developed a new version of the negative algorithm that uses real value representation. Elements of self/non-self space are points in the n-dimensional euclidean space ($\mathbb{R}^n$) while detectors are hyper-rectangles in $\mathbb{R}^n$. Detectors are evolved using a genetic algorithm that maximizes the covering of the non-self space while minimizing the matching self points. A niching technique is used in order to evolve multiple detectors that cover cooperatively the entire non-self space. Experiments are conducted on a subset of the DARPA99 data set and on a chaotic time series (Mackey-Glass).

Gonzalez and Dasgupta [77] combined AIS with a classification algorithm for anomaly detection. AIS is used for generating samples in the non-self space and the classification algorithm is executed using the self (normal) and the artificial non-self (abnormal) samples. Each detector, generated with the AIS, is used for generating artificial abnormal samples - points inside the detector. Experiments are conducted on a subset of the DARPA99 data set and with a chaotic time series (Mackey-Glass).

2.8 Data Sets Used in this Work

2.8.1 Machine Learning

Six benchmark data sets (publically available), were used as a testbed. See table 2.4.
Table 2.4: Machine Learning Testbed

<table>
<thead>
<tr>
<th>DATA SET</th>
<th>CLASSES</th>
<th>DIM</th>
<th>SAMPLES</th>
<th>TOTAL</th>
<th>PER CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAST</td>
<td>2</td>
<td>9</td>
<td>699</td>
<td>{458, 241}</td>
<td></td>
</tr>
<tr>
<td>PIMA</td>
<td>2</td>
<td>8</td>
<td>768</td>
<td>{500, 268}</td>
<td></td>
</tr>
<tr>
<td>HEART</td>
<td>2</td>
<td>13</td>
<td>270</td>
<td>{150, 120}</td>
<td></td>
</tr>
<tr>
<td>IRIS</td>
<td>3</td>
<td>4</td>
<td>150</td>
<td>{50, 50, 50}</td>
<td></td>
</tr>
<tr>
<td>WINE</td>
<td>3</td>
<td>13</td>
<td>178</td>
<td>{59, 71, 48}</td>
<td></td>
</tr>
<tr>
<td>GLASS</td>
<td>6</td>
<td>9</td>
<td>214</td>
<td>{70, 76, 17, 13, 9, 29}</td>
<td></td>
</tr>
</tbody>
</table>

2.8.2 10% KDDCup 99

This data set is 10% of the KDDCup99 data set [1] which is a version of the 1998 DARPA intrusion detection evaluation data set prepared and managed by MIT Lincoln Labs [2]. In this data set, 42 attributes (or fields) that usually characterize network traffic behavior compose each record. Some of these attributes are categorical (8) while others are discrete or numerical (34). The total number of records in the 10% data set is 492021.

This data set contains 22 different types of attacks that can be classified in four main intrusion classes, as is shown in table 2.5. The proportion of samples per class is not uniform; for example from class U2R the number of samples in the training data set is 59 while from class DOS the number of samples is 391458.

Table 2.5: Classes in 10% of the KDDCup 99

<table>
<thead>
<tr>
<th>CLASS</th>
<th>SUB-CLASSES</th>
<th>SAMPLES</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td></td>
<td>95278</td>
<td>19.3%</td>
</tr>
<tr>
<td>U2R</td>
<td>buffer_overflow, loadmodule, multitop, perl, rootkit</td>
<td>59</td>
<td>0.01%</td>
</tr>
<tr>
<td>R2L</td>
<td>ftp_write, guess_passwd, imap, phf, spy, warezclient, warezmaster</td>
<td>1119</td>
<td>0.23%</td>
</tr>
<tr>
<td>DOS</td>
<td>back, land, neptune, pod, smurf, teardrop</td>
<td>391458</td>
<td>79.5%</td>
</tr>
<tr>
<td>PRB</td>
<td>ipsweep, nmap, portsweep, satan</td>
<td>4107</td>
<td>0.83%</td>
</tr>
</tbody>
</table>

This data set was prepared in such a way that only the numerical attributes were used in the training-testing process. Moreover, a numerical attribute was removed since it was the same value for each record in the data set. Therefore, the reduced version of the KDDCup 99 data set contains 33 attributes.
2.8.3 Darpa 99

This data set was also obtained from the MIT Lincoln Lab [2]. It represents both normal and abnormal information since it was collected in a test network where simulated attacks occurred. The data set is composed of network traffic data (tcpdump, inside and outside network traffic), audit data (bsm), and file systems data. We used the outside tcpdump network data for a specific computer (e.g., hostname: marx), and then we applied the tool tcpstat to get traffic statistics. The first week’s data used for training was attack free. The second week’s data including some attacks was used for testing. In this work, network attacks were only considered.

Three parameters were selected to detect some specific type of attacks: bytes per second, packets per second, and ICMP packets per second. These parameters were sampled each minute using tcpstat and normalized. Because each parameter can be seen as a time series function, the features were extracted using a sliding overlapping window of size $n = 3$. Therefore, two sets of 9-dimensional feature vectors were generated; one as training data set, and the other as testing data set. The training data set is composed by 4000 normal data samples while the testing data set is composed by 5192 data samples, 56 of them being abnormal samples.
Chapter 3

Hybrid Adaptive Evolutionary
Algorithm (HAEA)

3.1 Introduction

Evolutionary algorithms (EA) are optimization techniques based on the principles of natural evolution [88]. Although EAs have been used successfully in solving optimization problems, the performance of this technique (measured in time consumed and solution quality) depends on the selection of the EA parameters. Moreover, the process of setting such parameters is considered a time-consuming task [125]. Several research works have tried to deal with this problem [44]. Some approaches tried to determine the appropriate parameter values by experimenting over a set of well-defined functions [37, 139], or by theoretical analysis [59, 10, 114]. Another set of approaches, called Parameter Adaptation (PA), tried to eliminate the parameter setting process by adapting parameters through the algorithm’s execution [36, 96, 154, 114, 147]. PA techniques can be roughly divided into centralized control techniques (central learning rule), decentralized control techniques, and hybrid control techniques.

In the centralized learning rule approach, genetic operator rates (such as mutation rate, crossover rate, etc) are adapted according to a global learning rule that takes into account the operator productivities through generations (iterations) [44]. Generally, only one operator is applied per generation, and it is selected based on its productivity. The productivity of an operator is measured in terms of good individuals produced by the operator. A good individual is one that improves the fitness measure of the current population. If an operator
generates a higher number of good individuals than other operators then its probability is rewarded. Two well
known centralized learning rule mechanism are the adaptive mechanism of Davis [36] and Julstrom [96].

In Davis’s approach, the operators and parents are stored for each offspring. If an offspring is better than
the current best individual in the population, the individual and the genetic operator used for generating it
are rewarded. Some portion of the reward is given recursively to the parents, grandparents, and so on. After
certain number of iterations the operator probabilities are updated according to the reward reached.

In the approach of Julstrom, a similar mechanism to Davis is developed, but only information about
recently generated chromosomes is maintained for each individual.

Although applied with relative success, a centralized technique has two main disadvantages: First, it re-
quires extra memory for storing information on the effect of each genetic operator applied to an individual,
parents, grandparents, etc. The amount of memory required grows exponentially on the number of genera-
tions used. For example, if the operator productivity is defined using the latest \( n \) generations and a single byte
is used per productivity information, the extra memory required will be approximated \( 2^n \) bytes per individual.
Second, a centralized technique requires an algorithm that uses such information for calculating the operator
productivity in a global sense; it cannot be defined at the individual level, only at the population level. There-
fore, the time complexity of the operator productivity grows linearly on the number of generations used and
is increased by the size of the population.

In decentralized control strategies, genetic operator rates are encoded in the individual and are subject to
the evolutionary process [44, 148]. Accordingly, genetic operator rates can be encoded as an array of real
values in the semi-open interval \([0.0,1.0)\), with the constraint that the sum of these values must be equal to
one [154]. Since the operator rates are encoded as real numbers, special genetic operators, meta-operators,
are applied to adapt or evolve them.

Although decentralized approaches have been applied to many problems with relative success, these
approaches have two main disadvantages: First, a set of meta genetic operators used for evolving the prob-
abilities must be defined. It is not easy to determine which operators to use and how these operators will
affect the evolution of the solution. Second, rates for such meta operators have to be given. Although these
techniques are not very sensitive to the setting of the meta operator rates, not every set of values works well.

In this chapter, a “hybrid” technique for parameter adaptation is proposed. Specifically, each individual
encodes its own operator rates and uses a randomized version of a learning rule mechanism for updating them.
Such a randomized learning rule mechanism is defined locally (per individual) and uses the productivity of
the genetic operator applied and a “random” generated learning rate. If a non-unary operator is selected,
the additional parents are chosen using a selection strategy. The operator rates are adapted according to
Algorithm 6 Hybrid Adaptive Evolutionary Algorithm (HAEA)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( t_0 = 0 )</td>
</tr>
<tr>
<td>2.</td>
<td>( P_0 = \text{initPopulation}(\lambda) )</td>
</tr>
<tr>
<td>3.</td>
<td>( \text{while}(\text{terminationCondition}(t, P_t) \text{ is false }) ) do</td>
</tr>
<tr>
<td>4.</td>
<td>( P_{t+1} = { } )</td>
</tr>
<tr>
<td>5.</td>
<td>for each ( \text{ind} \in P_t ) do</td>
</tr>
<tr>
<td>6.</td>
<td>( \text{rates} = \text{extract_rates}(\text{ind}) )</td>
</tr>
<tr>
<td>7.</td>
<td>( \delta = \text{random}(0,1) ) # learning rate</td>
</tr>
<tr>
<td>8.</td>
<td>( \text{oper} = \text{OP_SELECT}(\text{operators, rates}) )</td>
</tr>
<tr>
<td>9.</td>
<td>( \text{parents} = \text{PARENT_SELECTION}(P_t, \text{ind}) )</td>
</tr>
<tr>
<td>10.</td>
<td>( \text{offspring} = \text{apply}(\text{oper, parents}) )</td>
</tr>
<tr>
<td>11.</td>
<td>( \text{child} = \text{BEST}(\text{offspring, ind}) )</td>
</tr>
<tr>
<td>12.</td>
<td>if(\text{fitness(\text{child})} &gt; \text{fitness(\text{ind})}) ) then</td>
</tr>
<tr>
<td>13.</td>
<td>( \text{rates}[\text{oper}] = (1.0 + \delta)\times\text{rates}[\text{oper}] ) #reward</td>
</tr>
<tr>
<td>14.</td>
<td>else</td>
</tr>
<tr>
<td>15.</td>
<td>( \text{rates}[\text{oper}] = (1.0 - \delta)\times\text{rates}[\text{oper}] ) #punish</td>
</tr>
<tr>
<td>16.</td>
<td>( \text{normalize_rates}(\text{rates}) )</td>
</tr>
<tr>
<td>17.</td>
<td>( \text{set_rates}(\text{child, rates}) )</td>
</tr>
<tr>
<td>18.</td>
<td>( P_{t+1} = P_{t+1} \cup {\text{child}} )</td>
</tr>
<tr>
<td>19.</td>
<td>( t = t + 1 )</td>
</tr>
</tbody>
</table>

the performance achieved by the offspring compared to its parent, and the random learning rate generated.

Although the population size is an EA parameter that can be adapted, and each genetic operator has its own parameters that can be adapted, this work is only devoted to the adaptation of genetic operator rates.

This chapter is divided in six sections. Section 3.2 presents the proposed hybrid adaptive evolutionary algorithm. Section 3.3 reports some experimental results on binary encoding optimization problems. Section 3.4 reports some results on real encoding optimization problems. Section 3.5 introduces a niching technique and presents some experiments with multimodal functions. Section 3.6 draws some conclusions.

### 3.2 Hybrid Adaptive Control

Algorithm 6 presents the proposed Hybrid Adaptive Evolutionary Algorithm (HAEA). This algorithm is a mixture of ideas borrowed from Evolutionary Strategies (ES), decentralized control adaptation, and central control adaptation.

#### 3.2.1 Selection Mechanism

In HAEA, each individual is “independently” evolved from the other individuals of the population, as in evolutionary strategies [9]. In each generation, every individual selects only one operator from the set of possible operators (line 8). Such operator is selected according to the operator rates encoded into the individual.
a non-unary operator is applied, additional parents (the individual being evolved is considered a parent) are chosen according to any selection strategy, see line 9. As can be noticed, HAEA does not generate a parent population from which the next generation is totally produced. Among the offspring produced by the genetic operator, only one individual is chosen as child (line 11), and will take the place of its parent in the next population (line 17). In order to be able to preserve good individuals through evolution, HAEA compares the parent individual against the offspring generated by the operator. The BEST selection mechanism will determine the individual (parent or offspring) that has the highest fitness (line 11). Therefore, an individual is preserved through evolution if it is better than all the possible individuals generated by applying the genetic operator.

### 3.2.2 Encoding of Genetic Operator Rates

The genetic operator rates are encoded into the individual in the same way as decentralized control adaptation techniques, see figure 3.1. These probabilities are initialized (into the `initPopulation` method) with values following a uniform distribution $U[0, 1]$. A roulette selection scheme is used to select the operator to be applied (line 8). To do this, the operator rates are normalized in such a way that their summation is equal to one (line 16).

![Figure 3.1: Encoding of the operator probabilities in the chromosome](image)

### 3.2.3 Adapting the Probabilities

The performance of the child is compared against its parent performance in order to determine the productivity of the operator (lines 12-15). The operator is rewarded if the child is better than the parent and punished if it is worst. The magnitude of reward/punishment is defined by a learning rate that is randomly generated (line 7). Finally, operator rates are recomputed, normalized, and assigned to the individual that will be copied to the next population (lines 16-17). The learning rate is generated in a random fashion instead of setting it to a specific value for two main reasons. First, there is not a clear indication of the correct value that should be given for the learning rate; it can depend on the problem being solved. Second, several experiments encoding the learning rate into the chromosome [65] show that the behavior of the learning rate can be simulated with a random variable with uniform distribution.
3.2.4 Properties

Contrary to other adaptation techniques, HAEA does not try to determine and maintain an optimal rate for each genetic operator. Instead, HAEA tries to determine the appropriate operator rate at each instance in time according to the concrete conditions of the individuals. If the optimal solution is reached by an individual in some generation, then the rates of the individual will converge to the same value in subsequent generations. This is true because no genetic operator is able to improve the optimal solution, therefore any operator will be punished when applied and the other operators will be rewarded.

HAEA uses the same amount of extra information as a decentralized adaptive control; HAEA requires a matrix of $n \times M$ doubles, where $n$ is the number of different genetic operators and $M$ is the population size. Thus, the space complexity of HAEA is linear with respect to the number of operators (the population size is considered a constant). Also, the time expended in calculating and normalizing the operator rates is linear with respect to the number of operators $n \times M$ (lines 8 and 12-16). HAEA does not require special operators or additional parameter settings. Well known genetic operators can be used without any modification. Different schemes can be used in encoding the solution: binary, real, trees, programs, etc. The average fitness of the population grows monotonically iteration by iteration. One individual is always replaced by an individual with equal or higher fitness.

3.3 Experiments Using Binary Encoding

Experiments are conducted using binary encoding in order to determine the applicability of the proposed approach. In the binary encoding, the solution part of the problem is encoded as a string of binary symbols [88].

3.3.1 Genetic Operators

The performance reached by HAEA using different combinations of three well known genetic operators are compared: single point mutation, single point crossover, and a simple transposition.

- In the single bit mutation, one bit of the solution part is randomly selected (with uniform distribution) and flipped, see figure 3.2. Notice that this genetic operator always modifies the genome by changing only one single bit.

- In the simple transposition operator, two points in the solution part are randomly selected and the genes between such points are transposed [143], see figure 3.3.
In single point crossover, a cutting point in the solution part is randomly selected. Parents are divided in two parts (left and right) using such cutting point. The left part of one parent is combined with the right part of the other, see figure 3.4.

### 3.3.2 Test Functions

Four well known binary functions are used as a testbed, see table 3.1. The MaxOnes function is a very simple optimization problem. The fitness of an individual is defined as the number of its bits that are set to 1. Therefore, the solution of this problem is the individual that has all its bits set to 1. The 10 deceptive order-3 and 10 bounded deceptive order-4 functions were developed by Goldberg in 1989, for showing the type of functions that are hard for genetic algorithms (GAs) [60]. Goldberg defined a 3 bits (4 bits) function where the maximum is located between local minimas. Moreover, in the order-3 function there is a local maximum that is easily achieved by a GA. The 10 deceptive order-\(n\) functions are defined as concatenations of such \(n\) bits deceptive functions. The royal road function was developed by Forrest and Mitchell in order to show the kind of functions where crossover is usefull [49].
Table 3.1: Binary function tested

<table>
<thead>
<tr>
<th>Function</th>
<th>Chromosome Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Ones</td>
<td>100</td>
</tr>
<tr>
<td>Ten Deceptive Order-3 [60]</td>
<td>30 (3*10)</td>
</tr>
<tr>
<td>Ten Deceptive Order-4 [60]</td>
<td>40 (4*10)</td>
</tr>
<tr>
<td>Royal Road R1 [49]</td>
<td>64 (8*8)</td>
</tr>
</tbody>
</table>

3.3.3 Experimental Settings

For each test function, a population size of 100 individuals is used and the HAEA algorithm is executed a maximum of 10000 fitness function evaluations (100 generations). A tournament selection scheme of size 4 for determining the additional parent of crossover is applied. The reported results are the average over 100 different runs. Five different combinations of genetic operators are tested:

- Mutation alone (M),
- Mutation along crossover (MX),
- Mutation along transposition (MT),
- Crossover along transposition (XT), and
- Mutation along crossover and transposition (MXT).

Combinations M and MT are studied because these strategies can be considered as parallel hill climbing optimization processes, where the local search strategy is defined by the mutation and mutation-transposition operator, respectively.

3.3.4 Results

Table 3.2 shows the performance reached by HAEA on the binary functions. A value $a \pm b [c]$ indicates that a maximum average performance of $a$, with standard deviation of $b$, was reached by the HAEA using $c$ fitness evaluations in all the runs.
Table 3.2: Performance reached by HAE on the binary test functions using different set of operators.

<table>
<thead>
<tr>
<th></th>
<th>MaxOnes</th>
<th>Royal Road</th>
<th>Deceptive-3</th>
<th>Deceptive-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>90.00±1.32</td>
<td>19.12±4.37</td>
<td>292.34±1.44</td>
<td>34.14±0.81</td>
</tr>
<tr>
<td>MX</td>
<td>100.00±0.00</td>
<td>55.12±7.83</td>
<td>297.92±1.98</td>
<td>38.56±1.04</td>
</tr>
<tr>
<td>MT</td>
<td>84.56±1.69</td>
<td>19.60±4.44</td>
<td>291.72±1.44</td>
<td>33.64±0.79</td>
</tr>
<tr>
<td>XT</td>
<td>100.00±0.00</td>
<td>64.00±0.00</td>
<td>300.00±0.00</td>
<td>40.00±0.00</td>
</tr>
<tr>
<td>MXT</td>
<td>100.00±0.00</td>
<td>64.00±0.00</td>
<td>300.00±0.00</td>
<td>40.00±0.00</td>
</tr>
</tbody>
</table>

3.3.4.1 Genetic Operators Analysis

Figure 3.5 shows the evolution of the fitness value for each variation of HAE tested.

As expected, the performance of HAE is highly affected by the set of operators used. While the performance reached by HAE is low when using only mutation or mutation and transposition (M and MT), the performance is high and the optimal solution is found when all of the operators are used (MXT). These results indicate that HAE is able to recognize the usefulness of each genetic operator. When crossover and transposition were used (MXT and XT), HAE was able to find the optimal solution of each tested binary function. A possible explanation of this behavior is that transposition exploits the repetitive and symmetric structure definition of these test functions - any of these functions can be seen as the concatenation of small symmetric binary functions. Transposition can modify two of these small functions, the outer limits in the string being transposed, while maintaining other small functions (the inner part of the string being transposed). In this way, transposition assumes the role of a high level mutation. It is not surprising that performance and convergence rate reached by HAE when using crossover are higher than without using crossover. Moreover, HAE is not able to find the optimal solution for any of the four tested functions when
crossover is not included (M and MT). These results indicate that HAEA identifies the main role of crossover in the optimization process and allows it to exploit “good” regions of the search space.

Figure 3.6 presents the HAEA evolution of both solution and genetic operator rates for the royal road and deceptive tested functions when all three genetic operators were used (MXT). As shown, the probability of crossover increases rapidly in early stages of the evolutionary process. This means that crossover produced better individuals in early stages than other genetic operators. When, crossover was not enough for locating the optimal solution, its probability decreases allowing HAEA to try mutation and/or transposition. the general adaptation of operator rates, HAEA is not trying to determine and maintain an optimal probability for each genetic operator chromosome as other adaptation techniques. HAEA has temporal learning of operator rates. This adaptive property allows HAEA to have a higher chance to locate an optimal solution than another evolutionary algorithm. When the optimal solution was reached for all individuals in the population, the genetic operators probabilities converge to the same value. This behavior is expected since no genetic operator can produce a fitter individual than the optimal solution.

Figure 3.6: Performance (first row) and rates evolution (second row) using HAEA(MXT) on binary functions. (a) Royal Road, (b) Deceptive-3 and (c) Deceptive-4.
3.3.4.2 Comparing HAEA against other Evolutionary Algorithms and Previous Reported Results

Two standard GA are implemented in order to compare the performance of the proposed approach (HAEA): a Generational Genetic Algorithm (GGA) and a Steady State Genetic Algorithm (SSGA). The GGA used a tournament selection of size 4 for selecting each individual of the parent population, a standard bit mutation and single point crossover as genetic operators. The mutation rate of each bit in the chromosome is fixed to $\frac{1}{l}$ where $l$ is the length of the chromosome while the crossover rate is set to 0.7. These parameters are considered good for solving binary encoding problems [9, 125, 154, 114]. A tournament selection of size 4 for selecting the parent in the crossover operator, standard bit mutation and single point crossover as genetic operators, and kill-worst replacement policy are used in the SSGA implementation. The crossover rate is fixed to 1.0 while the mutation rate of each bit is set to $\frac{1}{l}$ where $l$ is the length of the chromosome. Table 3.3 summarizes the average performance reached by HAEA(MX), HAEA(MXT), the actual implementation of the GGA and SSGA, and compares them against the performance of some evolutionary algorithms reported in the literature. Results in rows 5-8 are reported by Tuson and Ross in [154]¹. Results in rows 9 and 10 are reported by Lobo in [114]². HAEA(MXT) and HAEA(MX) outperformed other EA approaches in the three hard binary functions: Royal Road, Deceptive-3 and Deceptive-4³. In the simple MaxOnes problem, HAEA found the global optimal solution, but it took around twice the number of evaluations required by SSGA based approaches and the regular GA used by Lobo.

Table 3.3: Comparison of HAEA performance on binary functions.

<table>
<thead>
<tr>
<th></th>
<th>MaxOnes</th>
<th>Royal Road</th>
<th>Deceptive-3</th>
<th>Deceptive-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAEA(MXT)</td>
<td>100.00±0.0[3900]</td>
<td>64.00±0.0[4900]</td>
<td>300.00±0.0[4800]</td>
<td>40.00±0.0[5000]</td>
</tr>
<tr>
<td>HAEA(MX)</td>
<td>100.00±0.0[3000]</td>
<td>55.12±7.83[10000]</td>
<td>297.92±1.98[2400]</td>
<td>38.56±1.04[9900]</td>
</tr>
<tr>
<td>GGA</td>
<td>100.00±0.0[4800]</td>
<td>49.52±9.24[10000]</td>
<td>293.52±3.01[10000]</td>
<td>37.08±1.29[2900]</td>
</tr>
<tr>
<td>SSGA</td>
<td>100.00±0.0[2800]</td>
<td>48.24±9.01[10000]</td>
<td>288.56±3.08[2000]</td>
<td>35.00±1.38[1100]</td>
</tr>
<tr>
<td>T-GGA [154]</td>
<td>99.96±0.2[7714]</td>
<td>35.52±6.02[7804]</td>
<td>289.68±2.41[5960]</td>
<td>-</td>
</tr>
<tr>
<td>T-SSGA [154]</td>
<td>100.00±0.0[2172]</td>
<td>40.64±7.65[3786]</td>
<td>289.12±3.08[3506]</td>
<td>-</td>
</tr>
<tr>
<td>T-D-GGA [154]</td>
<td>99.52±0.64[8438]</td>
<td>31.36±6.16[6086]</td>
<td>289.12±2.83[5306]</td>
<td>-</td>
</tr>
<tr>
<td>T-D-SSGA [154]</td>
<td>100.00±0.0[2791]</td>
<td>29.76±8.32[2428]</td>
<td>289.32±2.64[2355]</td>
<td>-</td>
</tr>
<tr>
<td>GA [114]</td>
<td>100.00±0.0[2500]</td>
<td>-</td>
<td>-</td>
<td>14.00[10000]</td>
</tr>
<tr>
<td>PL-GA [114]</td>
<td>100.00±0.0[7400]</td>
<td>-</td>
<td>-</td>
<td>28.00[10000]</td>
</tr>
</tbody>
</table>

¹These results are the average over 50 different runs. Row 5 presents the results obtained by Tuson and Ross using a Generational Genetic Algorithm (GGA) with fixed operator probabilities, row 6 using a Steady State Genetic Algorithm (SSGA), row 7 a GGA using distributed control parameter adaptation and row 8 a SSGA using distributed control. The population size used by Tuson and Ross was fixed to 100 individuals and the maximum number of iterations was set to 10000.

²These results are the average over 20 different runs. Row 9 presents the results obtained by Lobo using a regular GA with optimal parameters setting, population size of 100 individuals for the MaxOnes problem and 50 individuals for the deceptive-4 problem, while row 10 presents the results using the Parameter-less GA proposed by Lobo. Results for the deceptive 4 function were approximated from [114], figure 4.4.

³The Parameter-less GA was able to find the solution after 4e6 evaluations by increasing the population size.
3.4 Experiments Using Real Encoding

Experiments using real encoding are performed in order to determine the applicability of the proposed approach. In the real encoding, the solution part of the problem is encoded as a vector of real values [9].

3.4.1 Genetic Operators

Three different genetic operators are implemented: Gaussian mutation, Uniform mutation, and Single real point crossover.

- The Gaussian mutation operator adds to one component of the individual a random number $\Delta$ that follows a Gaussian distribution $G(0, \sigma)$. In our experiments we fixed $\sigma = \frac{\text{max} - \text{min}}{100}$, where $\text{max}$ and $\text{min}$ are the maximum and minimum values that can take the component.

- The uniform mutation operator replaces one of the component values of the individual with a random number following a uniform distribution $U(\text{min}, \text{max})$.

- The single point real crossover generates two individuals by randomly selecting one component $k$ and exchanging components $k, k+1, \ldots, n$ between the parent. For example let $p_1 = (3, 6, 5)$ and $p_2 = (1, 2, 8)$ be the parents, and $i = 2$ be the index of the component randomly selected. Then the two children generated are $c_1 = (3, 2, 8)$ and $c_2 = (1, 6, 5)$.

3.4.2 Test Functions

Experiments are conducted on the real functions shown in table 3.4. In the last three functions, the dimension of the problem is fixed to $n = 10$. For each test function, 64 bits are used to represent each real value.

Table 3.4: Real functions tested

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Feasible region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock</td>
<td>$f(x) = 100 \ast (x_1^2 - x_2) + (1 - x_1)^2$</td>
<td>$-2.048 \leq x_i \leq 2.048$</td>
</tr>
<tr>
<td>Schwefel</td>
<td>$f(x) = 418.9829 \ast n + \sum_{i=1}^{n} [-x_i \ast \sin(\sqrt{</td>
<td>x_i</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>$f(x) = n \ast A + \sum_{i=1}^{n} [x_i^2 - A \ast \cos(2\pi x_i)]$</td>
<td>$-5.12 \leq x_i \leq 5.12$</td>
</tr>
<tr>
<td>Griewangk</td>
<td>$f(x) = 1 + \sum_{i=1}^{n} \left[ \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) \right]$</td>
<td>$-600 \leq x_i \leq 600$</td>
</tr>
</tbody>
</table>

36
3.4.3 Experimental Settings

HAEA is executed a maximum of 20000 fitness evaluations (2000 iterations) with a population size of 100 individuals. The reported results are the average over 100 different runs. Four different combinations of genetic operators are used by HAEA:

- Gaussian and uniform mutations (GU),
- Crossover and Gaussian mutation (XG),
- Crossover and uniform mutation (XM), and
- Crossover, Uniform and Gaussian mutations (XUG).

3.4.4 Analysis of Results

Table 3.6 summarizes the results obtained by the proposed approach (HAEA) with different set of genetic operators after 20000 fitness evaluations. As expected, XUG has the best performance among the variations of HAEA tested (GU performs better than XUG only for the Rosenbrock function while XG performs better than XUG only for the Griewangk function).

<table>
<thead>
<tr>
<th></th>
<th>Rosenbrock</th>
<th>Schwefel</th>
<th>Rastrigin</th>
<th>Griewangk</th>
</tr>
</thead>
<tbody>
<tr>
<td>XUG</td>
<td>0.000509 ± 0.001013</td>
<td>0.005599 ± 0.011702</td>
<td>0.053614 ± 0.016808</td>
<td>0.054955 ± 0.029924</td>
</tr>
<tr>
<td>XU</td>
<td>0.004167 ± 0.004487</td>
<td>1.362088 ± 0.932791</td>
<td>0.240079 ± 0.155614</td>
<td>0.530857 ± 0.227458</td>
</tr>
<tr>
<td>XG</td>
<td>0.001322 ± 0.003630</td>
<td>140.5647 ± 123.7203</td>
<td>7.731156 ± 3.223595</td>
<td>0.050256 ± 0.025888</td>
</tr>
<tr>
<td>GU</td>
<td>0.000160 ± 0.000258</td>
<td>201.9162 ± 81.28619</td>
<td>6.320374 ± 1.462898</td>
<td>1.586373 ± 0.383703</td>
</tr>
</tbody>
</table>

Table 3.5: Solutions found by the tested EAs with the real functions

Figure 3.7 shows the fitness and operator rates evolution using HAEAXUG variation. The behavior of the operator rates is similar to the behavior observed for binary functions; HAEA applies with high probability the best operator in early stages of the evolution while at the end operators are applied with similar probabilities. Clearly, crossover is very useful in locating the optimal solution of the Rastrigin, Schwefel, and Griewangk functions, but it is not so good for the Rosenbrock saddle function. This behavior can be due to the fact that crossover generates good offspring that are located in the narrow local solution valley of the Rosenbrock saddle function. Due to the adaptation mechanism, an individual can leave or move faster from such narrow optimal solution by trying other genetic operators.
Figure 3.7: Performance evolution (first row) and Rates evolution (second row) using HAEA(MXT) on real functions. (a) Rosenbrock, (b) Schwefel, (c) Rastrigin

Two real-encoding versions of a generational algorithm (GGA(XG) and GGA(XU)) and two real-encoding versions of steady state algorithm (SSGA(XG) and SSGA(XU)) are implemented in order to compare the performance of the proposed approach (HAEA). Here the generational approaches use a tournament selection of size 4 for parent selection, with single point real crossover and Gaussian (Uniform) mutation as genetic operators. A tournament selection of size 4 for parent selection, with Gaussian (Uniform) mutation and single point real crossover as genetic operators, are used for the steady-state approaches with a worst-kill replacement policy – the worst individual of the population is replaced with the best offspring generated from the selected parent after crossover and possibly mutation). After applying different mutation and crossover rates, the best results obtained with 0.5 mutation rate and 0.7 crossover rate are reported. As can be noticed, both or none of the genetic operators may be applied to a single individual for both SSGA and GGA. Table 3.6 summarizes the results obtained by HAEA, our implementation of SSGA and GGA, and compares them against some results reported by Digalakis and Margaritis [42]4, and Patton et al [132]5.

4Digalakis and Margaritis varied the population size, the mutation rate and the crossover rate for a generational and a steady state genetic algorithm. The results reported in row 7 are the best results reported by Digalakis and Margaritis.
5Last row (Patton) reports the best result obtained by Patton et all [132] after 100000 evaluations.
Table 3.6: Solutions found by the tested EAs on real functions

<table>
<thead>
<tr>
<th>EA</th>
<th>Rosenbrock</th>
<th>Schwefel</th>
<th>Rastrigin</th>
<th>Griewangk</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAE(A)</td>
<td>0.00051±0.00101</td>
<td>0.00560±0.01170</td>
<td>0.05361±0.21681</td>
<td>0.05495±0.02992</td>
</tr>
<tr>
<td>GGA(X)</td>
<td>0.17278±0.11797</td>
<td>2.00096±1.21704</td>
<td>0.26500±0.15951</td>
<td>0.63355±0.24899</td>
</tr>
<tr>
<td>GGA(XG)</td>
<td>0.03852±0.03672</td>
<td>378.479±222.453</td>
<td>14.10895±5.01845</td>
<td>0.05074±0.02577</td>
</tr>
<tr>
<td>SSGA(XU)</td>
<td>0.06676±0.08754</td>
<td>0.88843±0.57802</td>
<td>0.12973±0.07862</td>
<td>0.32997±0.13091</td>
</tr>
<tr>
<td>SSGA(XG)</td>
<td>0.04842±0.04624</td>
<td>659.564±277.334</td>
<td>19.7102±7.80438</td>
<td>0.04772±0.02991</td>
</tr>
<tr>
<td>Digalakis [42]</td>
<td>0.40000000</td>
<td>-</td>
<td>10.000</td>
<td>0.7000</td>
</tr>
<tr>
<td>Patton [132]</td>
<td>-</td>
<td>-</td>
<td>4.8970</td>
<td>0.0043</td>
</tr>
</tbody>
</table>

As shown, HAEA compares well in solving real encoding optimization problems. Moreover, HAE(A) outperforms the SSGA and GGA approaches in all the tested functions. Variations of HAEA outperform the SSGA and GGA approaches in the Rosenbrock saddle function and in all the tested functions when Gaussian mutation is used. In general, the behavior of HAE(A) is similar to the behavior of SSGA(XU) and GGA(XU) for the Schwefel, Rastrigin and Griewangk functions.

3.5 Niching with HAEA

Nowadays, several optimization problems require not only one solution, but a set of good solutions. Because an EA will converge to only one solution, niching strategies have been developed for maintaining diversity in EAs [158]. A well known niching approach is deterministic crowding [117].

3.5.1 Deterministic Crowding (DC)

In DC, niches are formed and maintained by restricting the replacement of individuals [117]. The restriction is made in such a way that an individual is replaced with the most similar offspring generated. Therefore, DC requires a notion of similarity (or distance) between individuals. Algorithm 7 presents the DC algorithm.

Line 7, determines which offspring is closest to each parent; $d$ is a distance function. Lines 8-9 and 11-12 replaces the parents only if the fitness of the offspring is higher. Clearly, DC uses an elitist replacement strategy.

Although DC discovers the niches very fast, it is only able to maintain them for a few number of generations [118, 158]. In order to eliminate this problem, additional replacement policies can be introduced into the DC. For example, by adding a maximum distance ($\delta$) between parent and child for accepting the replacement in lines 8-9 and 11-12, see algorithm 8.
Algorithm 7 Deterministic Crowding

DC( P )
1. P = Shuffle( P )
2. P' = {}
3. N = size( P )
4. for i=1 to \( \frac{N}{2} \) do
5. \( \{c_1, c_2\} = \text{xover} ( P_i, P_{i+1} ) \)
6. \( \{c_1', c_2'\} = \text{OptionalMutation}\{c_1, c_2\} \)
7. if \( d(P_i, c_1') + d(P_{i+1}, c_2') \leq d(P_i, c_2') + d(P_{i+1}, c_1') \) then
8. if \( f(P_i) < f(c_1) \) then P' = P' \( \cup \) \{c_1\} else P' = P' \( \cup \) \{P_i\}
9. if \( f(P_{i+1}) < f(c_2) \) then P' = P' \( \cup \) \{c_2\} else P' = P' \( \cup \) \{P_{i+1}\}
10. else
11. if \( f(P_i) < f(c_2) \) then P' = P' \( \cup \) \{c_2\} else P' = P' \( \cup \) \{P_i\}
12. if \( f(P_{i+1}) < f(c_1) \) then P' = P' \( \cup \) \{c_1\} else P' = P' \( \cup \) \{P_{i+1}\}
13. return P'

Algorithm 8 Extended Deterministic Crowding

XDC( P )
1. P = Shuffle( P )

....
8. if \( f(P_i) < f(c_1) \) and \( d(P_i, c_1') \leq \delta \) then P' = P' \( \cup \) \{c_1'\} else P' = P' \( \cup \) \{P_i\}
9. if \( f(P_{i+1}) < f(c_2) \) and \( d(P_{i+1}, c_2') \leq \delta \) then P' = P' \( \cup \) \{c_2'\} else P' = P' \( \cup \) \{P_{i+1}\}
10. else
11. if \( f(P_i) < f(c_2) \) and \( d(P_i, c_2') \leq \delta \) then P' = P' \( \cup \) \{c_2'\} else P' = P' \( \cup \) \{P_i\}
12. if \( f(P_{i+1}) < f(c_1) \) and \( d(P_{i+1}, c_1') \leq \delta \) then P' = P' \( \cup \) \{c_1'\} else P' = P' \( \cup \) \{P_{i+1}\}
13. return P'
Algorithm 9 DC-HAEA

DC-HAEA( λ, terminationCondition )
1. \( P' = \{ \} \)
2. ...  
8. \( \text{child} = \text{Best}^*(\text{offspring}, \text{ind}) \)
9. \( x \)

\text{Best}^*(\text{offspring}, \text{ind})
1. \( N = \text{size} (\text{offspring}) \)
2. \( x = \text{ind} \)
3. \( \text{min} = 1e+8 \)
4. for \( i = 1 \) to \( N \) do
5. \( \text{if } d(\text{ind}, \text{offspring}_i) > 0 \text{ and } d(\text{ind}, \text{offspring}_i) < \text{min} \text{ then} \)
6. \( x = \text{offspring}_i \)
7. \( \text{min} = d(\text{ind}, \text{offspring}_i) \)
8. \( \text{if } f(\text{ind}) > f(x) \text{ then } x = \text{ind} \)
9. return \( x \)

3.5.2 Implementing DC in the HAEA

The implementation of a kind of DC in HAEA is straight-forward. The \text{BEST} selection strategy applied in line 8 is replaced with a selection strategy \text{Best}* that also includes the notion of distance, see algorithm 9. This modification allows one to use the notion of DC not only with crossover and mutation but with different genetic operators.

Lines 4-7, determine the closest offspring to the individual (excluding the parent itself). Line 8 selects the best between the parent and the closest individual (as lines 8-9 and 11-12 in DC algorithm). As the original DC, this version of DC maintains the niches for only a few generations when the crossover interaction problem arises. It is possible to include extra replacement policies in line 8 in order to maintain the niches for several more generations.

3.5.3 Experimentation

In order to determine the performance of the DC-HAEA, experiments are conducted in four well known one-dimensional multimodal functions [118], see figure 3.8.

M1 is a function with five peaks of the same height and width. These peaks are uniformly distributed in the \([0, 1]\) interval \((0.1, 0.3, 0.5, 0.7, 0.9)\). M2 is a function with five peaks of the same width but different height. These peaks are uniformly distributed in the \([0, 1]\) interval \((0.1, 0.3, 0.5, 0.7, 0.9)\). M3 is a function with five peaks of the same height but different width. These peaks are non-uniformly distributed in the \([0, 1]\) interval \((0.08, 0.25, 0.45, 0.68, 0.93)\). M4 is a function with five peaks of different height and width. These peaks are non-uniformly distributed in the \([0, 1]\) interval \((0.08, 0.25, 0.45, 0.68, 0.93)\).
3.5.3.1 Experimental Settings

HAEA is executed using a real encoding scheme, the tree genetic operators introduced in section 3.4.1, a population of 50 individuals, and a maximum of 200 generations. In order to maintain the niches for more generations, an individual is replaced with the offspring if the distance between both individuals is less than a predefined maximum distance. This distance is set to 0.1 in the experimentation carried on here. For consistency, the standard deviation of the Gauss mutation operator is set to 0.1. The reported results are the average over 50 runs.

3.5.3.2 Results Analysis

For all the test functions, DC-HAEA is able to find the niches very quickly (around 5 generations). Also, it is able to maintain them until the last generation. Table 3.7 shows the average best individual per niche after 200 generations.

Figure 3.8: Multimodal Functions
Table 3.7: Performance of DC-HAEA

<table>
<thead>
<tr>
<th>Niche</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.1003 ± 0.0008</td>
<td>0.3000 ± 0.0005</td>
<td>0.5901 ± 0.0006</td>
<td>0.6998 ± 0.0006</td>
<td>0.8998 ± 0.0008</td>
</tr>
<tr>
<td>M2</td>
<td>0.1000 ± 0.0001</td>
<td>0.2993 ± 0.0004</td>
<td>0.4986 ± 0.0005</td>
<td>0.6979 ± 0.0016</td>
<td>0.8977 ± 0.0012</td>
</tr>
<tr>
<td>M3</td>
<td>0.0798 ± 0.0012</td>
<td>0.2468 ± 0.0006</td>
<td>0.4505 ± 0.0006</td>
<td>0.6814 ± 0.0009</td>
<td>0.9336 ± 0.0010</td>
</tr>
<tr>
<td>M4</td>
<td>0.0797 ± 0.0001</td>
<td>0.2463 ± 0.0000</td>
<td>0.4494 ± 0.0004</td>
<td>0.6789 ± 0.0009</td>
<td>0.9298 ± 0.0014</td>
</tr>
</tbody>
</table>

DC-HAEA is able to find an individual very close to each local optima. The distance between the average solution found and the local optima is less than 0.001 in every case. Also, the behavior of DC-HAEA is similar in each run. The standard deviation between solution found in different is was less than 0.0015. Figure 3.9 shows the evolution of the population for the M4 function in a sample run.

After 40 iterations, all individuals in the population have converged to one of the local optima. After 200 iterations, the difference between individuals in the same niche is very small. Similar behavior is observed with the M1 to M3 functions.

3.6 Summary

In this chapter, a new evolutionary algorithm (HAEA) was introduced. HAEA evolves the operator rates at the same time it evolves the solution. HAEA was tested on several optimization functions, some of them using binary encoding while others using real encoding. The results indicated that HAEA is able to find good solutions for a variety of problems. Also a niching mechanism for HAEA was introduced. Such a mechanism is based on deterministic crowding (DC) [117]. Experiments with different multimodal functions have shown that DC-HAEA is able to find and maintain the desired niches.

This work uses HAEA for evolving fuzzy rules and fuzzy rule detectors (chapters 4 and 5). Each one of these approaches has its own parameters that should be tuned. The main goal of developing HAEA was to reduce the sensibility of such approaches to the EA process (parameter tuning).
Figure 3.9: Population evolution for M4 using DC-HAEA
Chapter 4

Evolution of Fuzzy Rule Classifiers

4.1 Introduction

Classification has received special attention of soft-computing techniques such as fuzzy Logic [91, 142, 74], neural networks [130, 82], and evolutionary algorithms [92, 57, 38]. In particular, fuzzy logic has been successfully applied in solving classification problems where boundaries between classes are not well defined [34, 12].

Due to high interpretability of crisp/fuzzy rule based classifiers (RBC) and the ability of evolutionary algorithms (EA) to find good solutions, some soft-computing research work has focused on developing evolutionary techniques for generating RBC [70, 34, 47, 92, 113, 74, 57, 38]. These techniques receive the name of Genetic Rule Based Systems (GRBS) [25, 26].

Due to the diversity of approaches for evolving fuzzy rules and the different experiments performed over them, it is difficult (and sometimes impossible) to compare them. The main purpose of the work presented in this chapter is to provide a general framework for evolving fuzzy rules and its application to solve the intrusion detection problem. This framework will allow the comparison of several approaches previously proposed in the literature with new ones.

Specifically, this chapter:

1. Presents a soft computing framework for solving classification problems, which combines fuzzy logic, evolutionary algorithms and class binarization. A classification problem, when needed, is divided in several two-class classification problems using a fuzzy version of a class binarization scheme. For each two-class classification problem, an evolutionary algorithm is used to evolve a fuzzy Rule Based
Classifier (RBC). Following the fuzzy class binarization scheme, the evolved fuzzy RBSs are combined to define the final fuzzy RBC.

2. Proposes a generic evolutionary algorithm for evolving fuzzy rules. In this way, an abstract fuzzy rule encoding scheme is developed. This encoding scheme covers several encoding mechanisms proposed in the literature. The concept of a fuzzy confusion matrix is introduced in order to define the fitness function. Special genetic operators for the above encoding are presented. The set of operators includes variable length crossover, gene addition, and gene deletion. Also, a disruption concept for fuzzy rule encodings is proposed in order to study the effect of genetic operators on different fuzzy rule encoding schemes.

3. Determines the performance of the proposed approach with different fuzzy logic operators, different encoding mechanisms, and different class binarization schemes. In this way, variations of approaches previously reported in the literature are implemented. These variations are designed in such a way that the original work spirit is preserved, and comparisons between all these works can be established.

4. Presents some preliminary results on solving the intrusion detection problem using the proposed framework.

4.2 Genetic Rule Based Systems (GRBS)

Genetic Rule Based Systems (GRBS) includes any technique that combines crisp/fuzzy rule based classifiers (RBC) and evolutionary algorithms (EA) for generating RBC [25, 26]. Of several GRBS approaches proposed, all of them differ from each other in at least one of the following aspects: number of rules that each individual encodes, type of rule expression encoded by an individual, and scope of the evolutionary process [70, 52, 25].

4.2.1 Pittsburgh and Michigan Approaches

According to the number of crisp/fuzzy rules that each individual of the population encodes, GFRBS can be divided in two broad approaches: Pittsburgh and Michigan. Each one has its advantages and disadvantages [52]. In the Pittsburgh approach, each individual of the population encodes a set of rules that will compose the RBC [39, 93, 146]. It is possible to capture the rules interaction in the fitness function but the search space grows exponentially with respect to the number of rules encoded. In the Michigan approach, each individual of the population encodes one rule [58, 18]. Although the search space is small compared with the Pittsburgh
approach, only one rule is encoded, it is not possible to capture the rule interactions in the fitness function [52].

Michigan approaches are further divided in two groups: simple and iterative learning. In the simple approach, all the rules are evolved using a single EA run. It is done by introducing a niching strategy in the EA [117, 61, 88]. In the iterative learning approach, the set of rules is evolved in several runs of an EA - a rule is evolved in each run [113, 92, 74]. The number of EA runs, the type of rule evolved and the mechanism used for combining such rules depends on the particular approach [52]. Some approaches penalize rules evolved in previous iterations and stop the iterative process when the set of rules is adequate [113, 74, 25, 27]. Other approaches run an EA as many times as the number of classes the problem has, each run with a different target class [34].

4.2.2 Rule Encoding

Although there are several crisp/fuzzy rule encoding mechanisms proposed for GRBS, only four of them are discussed here.

4.2.2.1 Conjunctions of Simple Terms

The condition length is fixed. It is composed by atomic expressions connected with a crisp/fuzzy and logic operator. Such atomic expressions are the only elements evolved [47, 92, 113, 74, 38].

4.2.2.2 Fixed Condition Structure

The condition is determined by a template where the logic operators and the tree structure are fixed. The atomic conditions are the only elements evolved [57].

4.2.2.3 Linear-tree Representation with Precedence of Operators

The tree structure of the condition is determined by priorities associated with each logic operator in the condition. Atomic expressions, logic operators and operator priorities are all evolved [34].

4.2.2.4 Evolution Scope

It is possible to use an EA for evolving (tuning) the fuzzy sets membership functions at the same time the fuzzy rule is evolved [92, 22, 26]. Murata [127] proposed a binary encoding for evolving fuzzy sets in such a way that a bit on '1' indicates when a fuzzy set membership has the value 1.0. The closest bits in on, at
the left and right side of such bit, indicate that the fuzzy set membership has value 0.0. Thus, fuzzy sets are encoded along with the fuzzy rule. Karr proposed a mechanism for evolving triangular fuzzy sets in. Karr [98] encoded into the chromosome the two control points that define the base of the triangular fuzzy set. The highest point, the point that will take the maximum fitness value of 1.0, is defined as the middle point between the evolved control points. Each control point is encoded independently using a set of \( m \) bits.

4.3 A Framework for Genetic Rule Based Systems

Evolving a GRBC involves the following four steps:

1. Collecting and preprocessing data from the target system. Collected data is labeled according to some criteria for defining data classes, noisy data samples (outliers) are removed, data samples with missing attribute values are completed, and normalizing numerical data attributes values. Usually, each value of a numerical attribute is normalized between 0.0 and 1.0 according to equation \( y = \frac{x - \text{min}}{\text{max} - \text{min}} \), where \( x \) is the numerical value, \( \text{min} \) is the minimum value for the attribute, and \( \text{max} \) is the maximum value.

2. Selecting a fuzzy class binarization technique and dividing the multi-class classification problem in several two-class classification problems accordingly. Such division includes the separation of the training data set in several training data sets. Each training data set includes specific data samples for the two class classification problem associated with it. If the original classification problem is a two-class problem, go directly to step 3.

3. Evolving a FRC for each two-class classification problem. It is possible to use the hybrid adaptive evolutionary algorithm (HAEA) proposed in chapter 3, one realization of the general encoding mechanism proposed in section 4.4.1, a set of special genetic operators like the operators proposed in this chapter, and the fitness function proposed in section 4.4.3.

4. Integrating the evolved fuzzy RBCs following the fuzzy class binarization scheme. These fuzzy RBCs will define the final fuzzy Rule Base System (RBS).

4.4 Evolution of Fuzzy Rules for a Two-class Problem

Figure 4.1 shows the evolutionary algorithm for evolving a fuzzy rule (two-class classification problem). This process takes as input the training data set (pre-processed to represent only two classes), applies the
evolutionary strategies, and returns one fuzzy rule. According to figure 4.1, three elements are important in
the definition of this evolutionary algorithm: rule encoding scheme, genetic operators, and fitness function.

Figure 4.1: Evolutionary Algorithm

4.4.1 Fuzzy Rule Encoding

Because an evolutionary algorithm is executed for each two-class problem, and a single fuzzy rule is the
classifier associated to it, it is not necessary to encode the class that the fuzzy rule represents (it is always
the positive class). Only the fuzzy expression that corresponds with the condition part of the fuzzy rule is
encoded. In this paper, only positive expressions are encoded into a chromosome.
4.4.1 Positive Expression (PE)

This type of expression receives the name of positive because the logic operator negation (*not*) is only allowed at the level of atomic expressions.

**Definition 4.1.** An expression is called **positive** if and only if it can be generated with the following production rules:

\[
\begin{align*}
ATOM & \rightarrow \text{var} \in \text{set} \mid \text{var} \notin \text{set} \\
PE & \rightarrow ATOM \mid (PE \land PE) \mid (PE \lor PE)
\end{align*}
\]

Despite this limitation, PEs allow the representation of very complex expressions. Moreover, it is possible to associate a tree to each PE by applying the following recursive definition:

\[
\begin{align*}
\text{Tree} (ATOM) & = [\lambda, ATOM, \lambda] \\
\text{Tree} (PE_1 \land PE_2) & = [\text{Tree} (PE_1), \land, \text{Tree} (PE_2)] \\
\text{Tree} (PE_1 \lor PE_2) & = [\text{Tree} (PE_1), \lor, \text{Tree} (PE_2)]
\end{align*}
\]

An interesting (and useful) property of PEs is presented in lemma 4.3. This property allows to define a general encoding scheme for PEs.

**Lemma 4.3.** Let \( R \) be a positive expression. Let \( n \geq 1 \) be the number of atomic expressions in \( R \). Then the number of logic operators in \( R \) is \( n-1 \).

**Proof.** Induction over the number of atomic expressions.

4.4.1.2 Encoding Function

From lemma 4.3, it is possible to use a linear structure (individual chromosome) for representing a PE. Such a structure can be defined as a list of genes. Each gene will carry an atomic expression, a logic operator if necessary, and extra decoding information (as is the case of priority trees [34]). Because the number of logic operators is one less than the number of atomic expressions (lemma 4.3), the logic operator encoded in the last gene is not taken into account. Figure 4.2 shows this general encoding scheme.

![Figure 4.2: General encoding scheme](image-url)
The number of bits used for encoding attributes $m$ attributes is $\lceil \log_2 (m) \rceil$. One bit can be used to represent the membership relation ($\in / \notin$) and one bit for the logic operator ($\land / \lor$). The number of bits used for encoding a fuzzy set depends on the scope of the evolutionary process; whether it is a fixed fuzzy space or it is using a fuzzy set tuning mechanism.

4.4.1.3 Fixed Fuzzy Space

For each fuzzy variable (independent and dependent variables), it is possible to predefine:

- The collection of fuzzy sets associated with the variable,
- The shape of each fuzzy set (membership function),
- The number of fuzzy sets, and
- The linguistic meaning of each fuzzy set (fuzzy set labels).

Because the training data set is usually normalized, for each numerical attribute it is possible to assign a fuzzy space as shown in figure 2.1. For non-numerical attributes, it can use the categorical values as crisp sets (fuzzy sets that do not overlap each other). Figure 4.3 shows the fuzzy space associated to an attribute with two categorical values (false, true).

![Figure 4.3: Categorical Fuzzy Sets](image)

If the number of fuzzy sets associated with each attribute is $m$, then $\lceil \log_2 (m) \rceil$ bits can be used to represent them.

4.4.1.4 Fuzzy Set Tuning

Instead of encoding the index of a predefined fuzzy set into each gene, a set of parameters defining a fuzzy set can be encoded into each gene and allow the EA to tune it. Isosceles triangular fuzzy sets can be tuned by
encoding two values defining the position of the base of the triangle \([98]\). Gaussian shaped fuzzy sets can be defined by encoding two values: the median and the standard deviation.

In this work, the fuzzy set tuning is restricted to trapezoidal fuzzy sets defined by two parameters. In this way, the attribute space is divided into \(m\) regions of the same length (\(m\) is a parameter given by the user). This division generates \(m + 1\) control points, see Figure 4.4.a. Given two control points, \(x\) and \(y\) (\(x \leq y\)), it is possible to define the trapezoidal fuzzy set: \((\max \{0, \frac{x-1}{m}\}, \frac{x}{m}, \frac{y}{m}, \min \{1, \frac{y+1}{m}\})\), see Figure 4.4.b\(^1\). Therefore, \(2 \lceil \log_2 (m + 1) \rceil\) bits are used for representing the two control points of the trapezoidal fuzzy set.

\[ \begin{align*}
1/m & \quad 2/m \quad \cdots \quad (m-1)/m \quad m/m \\
\hline
1.0 & \quad \hline
\end{align*} \]

(a) Division of the space in \(m\) intervals

\[ \begin{align*}
x/m & \quad \cdots \quad (x-1)/m \\
\hline
1.0 & \quad \hline
\end{align*} \]

(b) Two control points trapezoid

Figure 4.4: Tuning of Trapezoidal membership functions.

### 4.4.1.5 Extended Positive Normal Forms (XPNF)

It is possible to define an encoding function for PEs in normal form\(^2\). Moreover, an extension of this concept is introduced in [70]. This extension allows one to apply special genetic operators (gene addition and deletion) on fuzzy rules.

**Definition 4.2.** Let \(E\) be a PE and \(k \in \mathbb{N}^+\). \(E\) is in conjunctive normal form of order \(k\) (CNF-\(k\)) if and only if \(E = (A_{1,1} \lor \ldots \lor A_{1,k}) \land (A_{2,1} \lor \ldots \lor A_{2,k}) \land \ldots \land (A_{m,1} \lor \ldots \lor A_{m,k})\) for some \(m \in \mathbb{N}\) and \(A_{i,j}\) atomic expression.

The proposed extensions of these definitions are straightforward:

**Definition 4.3.** Let \(E\) be a PE and \(k \in \mathbb{N}^+\). \(E\) is in extended conjunctive normal form of order \(k\) (XCNF-\(k\)) if and only if \(E = (A_{1,1} \lor \ldots \lor A_{1,k}) \land (A_{2,1} \lor \ldots \lor A_{2,k}) \land \ldots \land (A_{m,1} \lor \ldots \lor A_{m,l})\) for some \(m, l \in \mathbb{N}\) and \(l \leq k\).

\(^1\)It will define a triangular fuzzy set instead of a trapezoidal if \(x = y\).

\(^2\)Some parenthesis are removed in order to simplify notation.
Disjunctive normal forms (\textbf{DNF}-\textit{k}) and the extended version (\textbf{XDNF}-\textit{k}) are dual definitions of \textbf{CNF}-\textit{k} and \textbf{XCNF}-\textit{k} respectively. Given the structure of normal forms, it is possible to encode them using the proposed general encoding but without using extra bits for logic operators nor for extra information. See figure 4.5.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textit{Gene}_1 & \ldots & \textit{Gene}_{n-1} & \textit{Gene}_n \\
\hline
\textit{Atom}_1 & \ldots & \textit{Atom}_{n-1} & \textit{Atom}_n \\
\hline
\textit{var} & \in / \notin & \textit{set} & \ldots \\
\hline
\end{tabular}
\caption{Conjunctive Normal Form (CNF)}
\end{table}

Normal form encoding schemes are defined in order to do a fair comparison with previous works. For example, \textbf{XCNF}-1 covers the encoding schemes proposed in [47, 92, 113, 74, 38]. Also, \textbf{XCNF}-\textit{k} and \textbf{XDNF}-\textit{k} (with \textit{k} \geq 2) are a good representative set of the encoding schemes proposed in [57].

\textbf{Lemma 4.4.} The time complexity of evaluating a fuzzy rule encoded using \textbf{XCNF}-\textit{k} or \textbf{XDNF}-\textit{k} is linear (\textit{O(n)}) with respect to the number of atomic expressions.

\textbf{Proof.} Evaluate from left to right the atomic expressions encoded in the individual and apply the \textit{and} fuzzy logic operator.

\subsection{Priority Trees}

This approach was proposed by Dasgupta and Gonzalez in [34]. For each logic operator a precedence value is given in the linear encoding. See figure 4.6. This precedence value indicates the order of evaluation; an operator with higher precedence value is evaluated first.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textit{Gene}_1 & \ldots & \textit{Gene}_{n-1} & \textit{Gene}_n \\
\hline
\textit{Atom}_1 & \ldots & \textit{Atom}_{n-1} & \textit{Atom}_n \\
\hline
\textit{var} & \in / \notin & \textit{set} & \ldots \\
\hline
\end{tabular}
\caption{Priority Trees}
\end{table}

\textbf{Lemma 4.5.} The time complexity of evaluating a fuzzy rule encoded using priority trees is linear (\textit{O(n)}) respectful to the number of atomic expressions.

\textbf{Proof.} Use the same mechanism applied by a compiler to evaluate arithmetic and logic expressions [4].

\subsection{Heaps (Complete Expression Trees)}

This approach was proposed in [70]. A heap tree is a binary tree that is filled completely on all the levels except possibly the last level that is filled from left to right [28], see figure 4.7.

The heap from a chromosome with \textit{n} genes, \( A = a_1a_2\ldots a_n \), can be obtained with the following process:
\( Tr(a_1a_2..a_n) = \begin{cases} 
\lambda, \text{atomic}(a_1), \lambda & \text{if } n = 1 \\
\text{rep}(Tr(a_1a_2..a_{n-1}), \text{atomic}(a_n), \text{oper}(a_n)) & \text{other case}
\end{cases} \) (4.3)

Here, \( \text{rep}(T, A, O) \) replaces the first leaf node of \( T \) (using the level tree enumeration [28]), with the node \([\text{first}_{\text{leaf}}(T), O, A]\). As can be noticed, it is necessary only to encode the atomic expressions along with the logic operators. No extra information is needed to recover the expression tree. See figure 4.8.

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Gene_1 & \ldots & Gene_{n-1} & Gene_n \\
\hline
Atom_1 & Op_1 & \ldots & Atom_{n-1} & Op_{n-1} & Atom_n & * \\
\hline
\end{tabular}
\end{center}
\end{table}

Lemma 4.6. The time complexity of evaluating a fuzzy rule encoded using heaps is linear \((O(n))\) respectful to the number of atomic expressions.

Proof. Follows the tree construction process given by equation 4.3. The implementation uses a circular queue.

4.4.2 Genetic Operators

In general, a genetic operator takes an individual (and additional parents if needed) and produces a set of offspring. These operators work at the genotype level and can produce offspring that do not resemble their parents (at the phenotype level). Because the disruption introduced by a genetic operator affects the convergence rate of an EA [88], this work introduces a notion of disruption for PEs using tree-structure similarity, see section 4.4.2.6. In this way, the disruption introduced by a genetic operator on different encoding mechanisms can be analyzed.
4.4.2.1 Variable Length Simple Point Crossover (VLSPX)

Given two chromosomes, \( A = a_1a_2..a_n \) and \( B = b_1b_2..b_m \), of size \( n \) and \( m \) respectively, the VLSPX selects a random point \( k \) in the interval \([2, \min\{n, m\} - 1]\), and generates two offspring \( C = a_1a_2..a_kb_{k+1}..b_m \) and \( D = b_1b_2..b_ka_{k+1}..a_n \). When PFEs are encoded, it is possible that VLSPX does not only exchange genes but modifies one of them (if the crossover point is selected in the middle of such gene). Figure 4.9 shows the effect of VLSPX on tree expressions when heaps encoding scheme is used.

![Figure 4.9: Effect of VLSPX on heaps encoding. (a) Parents. (b) Expression Trees Operation. (c) Offspring](image)

4.4.2.2 Single Bit Mutation (SBM)

Given a chromosome \( A = a_1a_2..a_k..a_n \), the SBM produces an offspring by flipping one random bit in it, \( C = a_1a_2..\overline{a_k}..a_n \). Figure 4.10 shows the effect of SBM using heaps encoding.

4.4.2.3 Gene Addition (ADD)

Given a chromosome \( A = a_1a_2..a_n \) of \( n \) genes (as explained in section 4.4.1.2), the ADD operator produces an offspring by generating a random gene \( r \) and appending it to the end of the chromosome: \( C = a_1a_2..a_nr \).
4.4.2.4 Gene Deletion (DEL)

Given a chromosome \( A = a_1a_2...a_n \) of \( n \geq 2 \) genes, the DEL operator produces an offspring by removing the last gene of the chromosome, \( C = a_1a_2...a_{n-1} \). Figure 4.11 shows the effect of ADD and DEL operators using heaps encoding.

![Image of DEL effect on heaps encoding](image)

Figure 4.11: Effect of DEL on heaps encoding. (a) Chromosomes. (b) Expression Trees.

It is possible to extend the DEL operator (\( XDEL \)) in order to randomly select a gene of the chromosome and remove such gene.

4.4.2.5 Gene Insertion (INS)

Given a chromosome \( A = a_1a_2...a_n \) of \( n \) genes (as explained in section 4.4.1.2), the INS operator produces an offspring by generating a random gene \( r \) and inserting it at a random position \( 0 \leq k \leq n \) into the chromosome: \( C = a_1a_2...a_kra_{k+1}...a_n \).
4.4.2.6 Disruption of PEs

A notion of distance between PEs is defined recursively using the tree structures associated with them:

\[
d(\lambda, E) = d(E, \lambda) = \text{size}(E)
\]

\[
d([L_1, M_1, R_1], [L_2, M_2, R_2]) = d(L_1, L_2) + d(R_1, R_2)
\]

This notion of distance captures the idea of tree-structure similarity. Notice that the tree information is not significant in the distance evaluation, but only the tree structure. Now a definition of disruption is straightforward.

**Definition 4.4.** Let \( E \) an PFE and \( \{E_i\}_{i=1, \ldots, n} \) the offspring generated by a genetic operator \( O \). The **disruption** generated by \( O \) is defined as:

\[
disr(O, E, \{E_i\}_{i=1, \ldots, n}) = \min\{d(E, E_i)\}_{i=1, \ldots, n}
\]

Clearly, the disruption of a genetic operator can change with the encoding scheme used. Lemma 4.7 determines the disruption introduced by the proposed genetic operators according to the encoding scheme used.

**Lemma 4.7.** Let \( E \) be a PFE with \( n \) atomic expressions. The maximum disruption generated by VLSPX, SBM, ADD, and DEL is:

1. \( \max = 2n - 1 \) when priority trees encoding scheme is used,
2. \( \max = 0 \) when heap encoding scheme is used, and
3. \( \max = 0 \) when XPNFs are used.

4.4.3 Fitness Function

The concept of fuzzy confusion matrix was introduced in [64] in order to determine the performance of an individual. This section proposes a simplification of such fitness function that has shown good performance in the classification problem.

4.4.3.1 Fuzzy Confusion Matrix

Values in a confusion matrix (see table 2.2) correspond with the cardinality of intersection sets. For example, \( PP \) is the number of positive samples that are classified (predicted) as positive, i.e., the cardinality of the set:
Actual-Positive \cap Predicted-Positive. These values can be calculated by using the membership function of
the data samples to the actual and predicted data sets as:

\[
PP = \sum_{i=1}^{n} \mu_A(d_i) \land \mu_B(d_i) \quad (4.6)
\]

\[
PN = \sum_{i=1}^{n} \mu_C(d_i) \land \mu_B(d_i) = \sum_{i=1}^{n} \mu_A(d_i) \land \mu_B(d_i) \quad (4.7)
\]

\[
NP = \sum_{i=1}^{n} \mu_A(d_i) \land \mu_D(d_i) = \sum_{i=1}^{n} \mu_A(d_i) \land \mu_B(d_i) \quad (4.8)
\]

\[
NN = \sum_{i=1}^{n} \mu_C(d_i) \land \mu_D(d_i) = \sum_{i=1}^{n} \mu_A(d_i) \land \mu_B(d_i) \quad (4.9)
\]

Where,

- \( n \) is the number of samples used to test the classifier,
- \( A \) is the actual (real) positive set,
- \( B \) is the predicted positive set,
- \( C \) is the actual negative set,
- \( D \) is the predicted negative set, and
- \( d_i \) is the \( i \)-th data record sample in the data set.

Notice that, for a two-class classification problem, one only needs to know the membership of a data sample
to the actual data set and to the predicted membership value of the positive set.

Equations 4.6, 4.7, 4.8 and 4.9 can be extended to fuzzy sets and fuzzy rules. The degree of membership
of the data sample to the actual positive set is given by the data sample label and the predicted membership
to the positive set is calculated as the truth-value of the condition part of the fuzzy rule. The confusion
matrix generated by using these extensions is called fuzzy confusion matrix. Performance metrics in table
2.3 can be generated from the fuzzy confusion matrix. Such new performance metrics will be called fuzzy
performance metrics: (fuzzy accuracy, fuzzy true positives, etc ).
4.4.3.2 Definition

Since the goal of the evolutionary process is to generate a simple fuzzy rule that can discriminate the positive class from the negative, the fitness of an individual is defined by the fuzzy accuracy ($FAC$), and the number of atomic conditions defining the fuzzy rule - fuzzy rule length ($FRL$). In this way, the optimization problem is a two-goal objective function: maximizing the FAC while minimizing the fuzzy rule length (FRL). Although there are several ways to deal with multi-goal objective functions, in this work, the weighted sum technique was used. Therefore, the fitness of an individual is calculated using equation 4.10.

$$
fitness(R) = w \times FAC(R) + (1 - w) \times \left(1 - \frac{FRL(R)}{M}\right)
$$

(4.10)

Here, $w$ is the weight associated with the fuzzy accuracy reached by the individual and $M$ is the maximum number of atomic expressions defining a fuzzy rule.

4.4.4 Rule Extraction

The best individual of the population, according to the fitness value, will determine the fuzzy rule that will be used for discriminating between the two classes under consideration.

4.4.5 Experiments with Machine Learning Two-class Problems

Experiments are conducted on the machine learning two-class classification problems defined in section 2.8: Breast, Heart and Pima.

4.4.5.1 Experimental Settings

Different combinations of encoding schemes with and without tuning of fuzzy sets and fuzzy logic operators are studied. For each one of these combinations, the two-class evolutionary algorithm (which evolves the fuzzy rule associated with each two-class problem) is executed for 100 iterations using 100 individuals as population and VLSPX, SBB, ADD and DEL as genetic operators. Individuals of the initial population are randomly generated with a length varying between 1 and the number of attributes defining the data set (9 for Breast, 8 for Pima and 13 for Heart). A 10 fold cross-validation technique is applied to each data set 5 different times. The reported results are the average over those 50 runs.
4.4.5.2 Comparison of Encoding Schemes

Eight different encoding schemes are tested in order to cover as many encoding schemes as proposed previously in the literature. These encodings are: Heaps or complete expression trees (HEAP), Priority Trees with 3 bits for the precedence value (PTREE), Extended Disjunctive (Conjunctive) Normal Forms of length 1, 2, and 3 (XDNF-1, XDNF-2, XDNF-3, XCNF-1, XCNF-2, and XCNF-3). These encoding mechanisms are used without fuzzy set tuning and with max-or and min-and fuzzy logic operators in order to determine the real effect of such encoding. Five fuzzy sets, as shown in figure 2.1, define the fuzzy space of each attribute.

Table 4.1 shows the performance (accuracy) of these encoding mechanisms on the test-bed. A value of \(a \pm b (c)\) indicates that the encoding reached an accuracy of \(a\%\) with an standard deviation of \(b\%\) and it was ranked \(c\) among all different encodings. The final ranking of an encoding strategy is given by the summation of the accuracy reached on all the data sets Therefore, a higher summation implies a higher rank.

<table>
<thead>
<tr>
<th></th>
<th>BREAST</th>
<th>HEART</th>
<th>PIMA</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEAP</td>
<td>94.94±2.69 (3)</td>
<td>78.67±8.30 (1)</td>
<td>73.35±6.00 (7)</td>
<td>246.96 (1)</td>
</tr>
<tr>
<td>PTREE</td>
<td>95.05±2.71 (1)</td>
<td>77.55±7.53 (2)</td>
<td>72.98±5.92 (8)</td>
<td>245.58 (2)</td>
</tr>
<tr>
<td>XDNF-1</td>
<td>95.02±2.68 (2)</td>
<td>75.33±6.00 (4)</td>
<td>73.58±5.56 (6)</td>
<td>243.93 (4)</td>
</tr>
<tr>
<td>XDNF-2</td>
<td>90.76±5.56 (7)</td>
<td>74.74±8.56 (6)</td>
<td>73.73±5.49 (5)</td>
<td>239.23 (7)</td>
</tr>
<tr>
<td>XDNF-3</td>
<td>87.61±5.31 (8)</td>
<td>72.37±5.80 (8)</td>
<td>73.76±5.90 (4)</td>
<td>233.74 (8)</td>
</tr>
<tr>
<td>XCNF-1</td>
<td>92.71±3.02 (6)</td>
<td>73.85±6.43 (7)</td>
<td>73.87±5.80 (3)</td>
<td>240.43 (6)</td>
</tr>
<tr>
<td>XCNF-2</td>
<td>93.43±4.70 (5)</td>
<td>74.74±5.62 (5)</td>
<td>74.15±5.56 (1)</td>
<td>242.32 (5)</td>
</tr>
<tr>
<td>XCNF-3</td>
<td>94.37±3.66 (4)</td>
<td>75.56±6.71 (3)</td>
<td>94.37±3.66 (4)</td>
<td>244.08 (3)</td>
</tr>
</tbody>
</table>

In general, the performance reached with Heaps encoding mechanism is better than the performance reached by other encoding mechanisms. Heaps encoding (first row) is ranked first on the Heart data set and third in the Breast data set. Although ranked seventh on the Pima data set, the difference on the performance reached by heaps encoding and the winner encoding XCNF-2 (seventh row) is lower than 1%. Moreover, XCNF-2 is ranked first for the Pima data set, but it is ranked fifth in the Heart and Breast data sets with a difference on the performance higher than 4% and 1% compared to the winner encoding. Similar behavior is observed on all the extended normal forms. These results suggest that the performance reached by any XDFN-k or XCNF-k encoding is very sensitive to the type of data set being mined. Finally, Priority-Trees encoding (second row) is ranked second, showing a consistent performance and similar to the heaps encoding.
4.4.5.3 Comparison of Fuzzy Logic Operators

Different fuzzy logic operators are compared in order to determine their effect in the performance reached by the proposed approach. In this way, the proposed approach is tested using each of the possible six combinations of the fuzzy logic operators shown in Table 2.1, heaps encoding scheme and a fixed set of fuzzy sets per attribute (no fuzzy set tuning). Table 4.2 shows the performance reached by these combinations.

| Table 4.2: Performance of different sets of fuzzy logic operators |
|---------------------------------|-----------------|-----------------|-----------------|-----------------|
|                                | BREAST          | HEART           | PIMA            | RANK            |
| Min-Max                        | 94.94±2.69 (3)  | 78.67±8.30 (1)  | 73.35±6.00 (7)  | 246.96 (1)      |
| Min-Rest. Sum                  | 94.54±3.00 (6)  | 78.07±7.51 (6)  | 73.61±5.92 (4)  | 246.22 (6)      |
| Prod.-Max                      | 95.14±2.47 (1)  | 78.15±8.06 (5)  | 73.66±5.83 (3)  | 246.95 (4)      |
| Prod.-Rest. Sum                | 94.71±2.84 (4)  | 78.67±7.65 (2)  | 73.34±5.66 (6)  | 246.72 (5)      |
| Avg-Max                        | 94.94±2.71 (3)  | 78.44±7.79 (4)  | 73.79±5.75 (1)  | 247.17 (2)      |
| Avg-Rest. Sum                  | 94.62±3.00 (5)  | 79.41±7.30 (1)  | 73.68±5.85 (2)  | 247.71 (1)      |

Notice that, the performance of the proposed approach is not strongly affected for selecting different fuzzy logic operators. The maximum variation between the performance reached by using any of these operators is less than 1.35% (Heart data set). These results indicate that the best fuzzy-and operator for evolving fuzzy rules is the Average-And operator. Combinations including this operator are ranked first and second. However, there is no clear indication about which is the best fuzzy-or operator for evolving fuzzy rules. In one hand, Restricted-Sum-Or performs very well along Average-And (ranked first) but it does not perform so well when combined with other fuzzy-and operators (ranked fifth and sixth). On the other hand, the performance reached when Max-Or is used is very consistent among the different fuzzy-and operators tested being ranked second, third and fourth. Therefore, the best combination of fuzzy logic operators is Average-And with Maximum-Or.

4.4.5.4 Fuzzy Sets Tuning

In order to compare tuning of fuzzy sets against predefined fuzzy sets, the proposed approach is executed using Heaps encoding, Average-And:Maximum-Or as fuzzy logic operators and tuning of fuzzy sets using 6 divisions. This number of divisions is selected in order to match the division generated using the 5 predefined fuzzy sets used in previous sections. Table 4.3 shows the performance of the fixed and tuning fuzzy sets approach.

As shown, the performance reached by the proposed approach using tuning fuzzy sets is similar to the performance reached using a predefined collection of fuzzy sets, in some cases is better (Pima and Heart
Table 4.3: Performance of fuzzy set tuning against fixed fuzzy sets.

<table>
<thead>
<tr>
<th></th>
<th>PRE-DEFINED</th>
<th>TUNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAST</td>
<td>94.94±2.71</td>
<td>94.85±2.41</td>
</tr>
<tr>
<td>HEART</td>
<td>78.44±7.79</td>
<td>78.96±8.26</td>
</tr>
<tr>
<td>PIMA</td>
<td>73.79±5.75</td>
<td>74.21±5.63</td>
</tr>
</tbody>
</table>

data sets). These results indicate that the tuning mechanism can evolve fuzzy sets that approximate patterns hidden in the data set. Take for example the following fuzzy rule generated for the Pima data set in a sample run:

\[
\text{IF } x_8 \text{ is set}_{2,3} \text{ AND } x_1 \text{ is set}_{0,1} \text{ OR } x_2 \text{ is not set}_{0,4} \text{ THEN Diabetes-Disease}
\]

Here, \(set_{x,y}\) represents the trapezoidal fuzzy set \((\max\{0, \frac{x-1}{m}\}, \frac{x}{m}, \frac{y}{m}, \min\{1, \frac{y+1}{m}\})\) with \(m\) being the number of divisions. In order to approximate the atomic expression \(x_8 \text{ is set}_{2,3}\) using a collection of predefined fuzzy sets, it will require the Restricted-Sum Or fuzzy logic operator and the condition \(x_8 \text{ is ML OR } x_8 \text{ is } M\).

### 4.4.6 Fuzzy Rule Complexity

Figure 4.12 shows the evolution of the fuzzy rule length for both the best individual in the population and the average length of individuals in the population. Clearly, T-FD produces simple fuzzy rules as no more than 4 attributes are included in the condition part of the fuzzy rules.
Algorithm 10 Fuzzy Unordered Classification

CLASSIFY(classifier[1..m], sample)
1. winners = ∅
2. for i = 1 to m do
3.   winners = winners ∪ {μ_{positive}(classifier_i, sample)}
4. return DEFUZZY(winners)

4.5 Fuzzy Class Binarization

4.5.1 Fuzzy Unordered

The unordered class binarization strategy is fuzzyfied by keeping in the winners set the membership of all the classifiers and then applying a defuzzyfication technique instead of the PICK_ONE function, see algorithm 2. Algorithm 10 presents the proposed fuzzy version of unordered class binarization. This binarization scheme has been successfully applied with a maximum defuzzyfication technique in [70, 64, 34].

The following lemma determines the training complexity of a linear algorithm that uses fuzzy unordered binarization.

Lemma 4.1. Let n be the size of the full training data set. If the time complexity of the two-class learning algorithm is linear with respect to the size of the training data set $O(n)$, then the time complexity of the learning algorithm using fuzzy unordered binarization is $O(nm)$ where m is the number of classes.

Proof. The $i$-th iteration of the fuzzy binarization learning process uses the full data set, labeling some data samples as positive (samples of class $i$), and the remaining as negative. Then, each iteration has time complexity of $O(n) = cn$ for some constant $c > 0$. The process is repeated as many classes has the problem, i.e. $m$ times. Therefore, the time complexity is $m * O(n) = mcn = O(mn)$.

4.5.2 Fuzzy Round Robin

It is possible to create a fuzzy version of the round robin binarization scheme. Instead of doing a crisp voting accumulation (lines 5-8), it is possible to do a fuzzy voting accumulation (using the membership value generated by the fuzzy classifiers), normalizing these values, and applying a defuzzyfication technique. Algorithm 11 shows the proposed fuzzy round robin.

The following lemma determines the training complexity of a linear algorithm that uses fuzzy round robin binarization.

Lemma 4.2. Let n be the size of the full training data set. If the time complexity of the two-class learning algorithm is linear with respect to the size of the training data set $O(n)$, then the time complexity of the learning algorithm using fuzzy unordered binarization is $O(nm)$ where $m$ is the number of classes.
Algorithm 11 Fuzzy round robin classification

CLASSIFY( classifier[1..m − 1][i + 1..m], sample )
1. for i = 1 to m do
2. vote[i] = 0
3. for i = 1 to m − 1 do
4. for j = i + 1 to m do
5. vote[i] = vote[i] + µi( classifier[i][j], sample )
6. return DEFUZZY( NORMALIZE(vote) )

Proof. The i,j-th iteration of the fuzzy binarization learning process uses a data set composed only by samples of class i as positive and samples of class j as negative. Let nk be the number of samples of class k in the original training data set. Then, each iteration has time complexity of \( O(n_i + n_j) = c(n_i + n_j) \) for some constant \( c > 0 \). The total time expended by the process is \( \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} c(n_i + n_j) = cmn \). Therefore the time complexity is \( O(mn) \).

4.5.3 Experiments with Multi-Class Problems

4.5.3.1 Experimental Settings

Experiments are performed in the multi-class machine learning data sets introduced in chapter 2. Both fuzzy class binarizations (unordered and round robin) are tested using Heaps encoding scheme, tuning of fuzzy sets, and Average-Restricted Sum as fuzzy logic operators. The two-class evolutionary algorithm (which evolves the fuzzy rule associated with each two-class problem) was executed for 100 iterations using 100 individuals as population and VLSPX, SBM, ADD and DEL as genetic operators. A 10 folding cross-validation technique was applied to each data set 5 different times. The reported results are the average over those 50 runs.

4.5.3.2 Results

Table 4.4 compares the performance reached by fuzzy unordered class binarization (FUCB) against the one reached by fuzzy round robin (FRRCB). A value \( a \pm b \) indicates that the approach reaches an average accuracy of \( a \% \) with a standard deviation of \( b \% \).

According to this results, there is no binarization technique that can be considered the best for any data set. In some cases, FUBC outperforms FRRCB (Iris data set) while FRRBC outperforms FUBC in other cases (Glass data set).
Table 4.4: Performance of fuzzy class binarizations.

<table>
<thead>
<tr>
<th></th>
<th>FUCB</th>
<th>FRR CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIS</td>
<td>96.13±4.24</td>
<td>95.06±5.30</td>
</tr>
<tr>
<td>WINE</td>
<td>94.19±5.22</td>
<td>94.39±5.83</td>
</tr>
<tr>
<td>GLASS</td>
<td>53.66±9.26</td>
<td>53.82±9.07</td>
</tr>
</tbody>
</table>

4.6 Comparison with Results Reported in the Literature

The results produced by the proposed approach, Evolution of Fuzzy Rules (EFR), using the best fuzzy logic operator combination (Max-Or with Average-And), heap encoding and tuning of fuzzy sets are compared against results previously reported in the literature. See table 5.2. As shown, these results (first and second row) are similar to those reported in the literature.

Table 4.5: Comparative performance of the proposed approach

<table>
<thead>
<tr>
<th>Method</th>
<th>BREAST</th>
<th>PIMA</th>
<th>HEART</th>
<th>WINE</th>
<th>GLASS</th>
<th>Statistical Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFR-FUCB</td>
<td>94.85</td>
<td>74.21</td>
<td>78.96</td>
<td>94.19</td>
<td>53.66</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>EFR-FRR CB</td>
<td>94.85</td>
<td>74.21</td>
<td>78.96</td>
<td>94.39</td>
<td>53.82</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>QDA</td>
<td>94.90</td>
<td>74.80</td>
<td>57.80</td>
<td>99.40</td>
<td>-</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>LDA</td>
<td>96.00</td>
<td>77.20</td>
<td>60.40</td>
<td>98.90</td>
<td>59.70</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>GAP.sel</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>97.20</td>
<td>70.10</td>
<td>None</td>
</tr>
<tr>
<td>GAP.par</td>
<td>-</td>
<td>-</td>
<td>93.30</td>
<td>67.30</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>C4.5</td>
<td>94.70</td>
<td>73.00</td>
<td>22.90</td>
<td>-</td>
<td>68.20</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>kNN</td>
<td>96.90</td>
<td>71.90</td>
<td>65.60</td>
<td>95.50</td>
<td>72.00</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>SSV</td>
<td>96.30</td>
<td>73.70</td>
<td>-</td>
<td>98.30</td>
<td>-</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>FSM</td>
<td>96.90</td>
<td>-</td>
<td>-</td>
<td>96.10</td>
<td>-</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>WM</td>
<td>87.10</td>
<td>71.30</td>
<td>-</td>
<td>-</td>
<td>54.70</td>
<td>-</td>
</tr>
<tr>
<td>GIL</td>
<td>90.10</td>
<td>73.10</td>
<td>-</td>
<td>-</td>
<td>64.60</td>
<td>-</td>
</tr>
<tr>
<td>ABD</td>
<td>96.00</td>
<td>75.90</td>
<td>-</td>
<td>-</td>
<td>64.10</td>
<td>-</td>
</tr>
<tr>
<td>ABA</td>
<td>95.10</td>
<td>74.80</td>
<td>-</td>
<td>-</td>
<td>64.10</td>
<td>random (50-50)%</td>
</tr>
</tbody>
</table>

4.7 Experiments with the KDDCup99 Intrusion Detection Data Set

4.7.1 Experimental Setting

A small portion (less than 1%) of the KDDCup99 data set was selected and used as training data set while the remaining part of the data set was used as testing data. The training data set was selected in a random fashion.

---

3 Results reported for QDA, LDA, C4.5, kNN, SSV and FSM taken from [159]. Results for WM, GIL, ABD, and ABA taken from [85]. Results for GAP taken from [127], where the number of fuzzy rules was close to the number of classes.

4 Although all of these results were obtained with different statistical validation methods (leave-one-out, or 10-cross-validation) or not statistical validation, the values reported here are an indicative of the performance of our proposed approach.
with the distribution shown in table 4.6. The score of the trained classifier is calculated as the classification accuracy over the testing set. This process was repeated 10 times and the reported results are the average of these 10 runs.

<table>
<thead>
<tr>
<th>CLASS</th>
<th>SAMPLES</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>2000</td>
<td>2.1%</td>
</tr>
<tr>
<td>DOS</td>
<td>1500</td>
<td>0.38%</td>
</tr>
<tr>
<td>PRB</td>
<td>411</td>
<td>10%</td>
</tr>
<tr>
<td>R2L</td>
<td>112</td>
<td>10%</td>
</tr>
<tr>
<td>U2R</td>
<td>30</td>
<td>50%</td>
</tr>
</tbody>
</table>

The results reported in this section are obtained by considering the IDP as a two-class problem and using Heaps encoding, a population size of 100 individuals, a maximum of 100 iterations, *Average-And* and *Restricted-Sum-Or* fuzzy logic operators, and fuzzy set tuning mechanism.

### 4.7.2 Analysis of Results

When the KDDCup problem is considered as a two-class problem, EFR is able to generate a compact characterization (a single fuzzy rule) of the normal behavior. The following is a fuzzy rule generated in a sample run:

\[
\text{IF } \text{dst_host_same_src_port_rate is not set0.3 OR srv_error_rate is set1.4 OR same_svr_rate is set0.4} \\
\text{THEN data_record is NORMAL}
\]

Only three attributes are considered in the condition part of the fuzzy rule. Moreover the average length of the extracted fuzzy rule was lower than 5 atomic expressions. Clearly, EFR is doing feature selection during the training process.

EFR generates the average ROC curve shown in Figure 4.13. According to this ROC curve, EFR is able to detect 96.37% of the attacks while it generates a false alarm rate of 8.4% when the minimum value for fire the normal rule is set to 0.5.

When considering each attack independently, EFR generates the ROC curves shown in Figure 4.14.

As expected, the behavior of the ROC curve considering only DoS attack is almost the same as considering all the attacks. It is due to the fact that DoS includes more than 99% of the attacks. Notice that the detection of other attacks in not so bad taking into account the small proportion of samples belonging to those attacks. For example, EFR is able to detect 89.45% of the Prb attacks. Although the worst attack detection is for U2R
Figure 4.13: ROC Curve generated by EFR for the KDDCup data set.

with a 55.77% detected, this behavior is not strange since the number of samples of this class in the training data set (and in the full data set) is less than 100.

The classification accuracy of EFR is shown in table 4.7 along with the performance achieved by the winner group in the KDDCup 99 contest [3]. Although similar, these results are not comparable, as different data sets were used for each technique. Hence, this information is for reference only.

<table>
<thead>
<tr>
<th>Class</th>
<th>EFR</th>
<th>Winner Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>91.60%</td>
<td>94.50%</td>
</tr>
<tr>
<td>U2R</td>
<td>55.77%</td>
<td>13.2%</td>
</tr>
<tr>
<td>R2L</td>
<td>66.45%</td>
<td>8.4%</td>
</tr>
<tr>
<td>DOS</td>
<td>96.52%</td>
<td>97.10%</td>
</tr>
<tr>
<td>PRB</td>
<td>89.45%</td>
<td>83.30%</td>
</tr>
</tbody>
</table>

4.8 Summary

A framework for evolving fuzzy rules was proposed. The proposed mechanism follows an iterative Michigan approach using a fuzzy class binarization scheme. Several encoding schemes were compared and the results were reported. As expected the proposed framework provides a simple mechanism for determining the advantages and disadvantages of a particular encoding scheme, genetic operator, fuzzy sets tuning mech-
Figure 4.14: ROC Curve generated by EFR for each different attack.

anism and/or class binarization strategy. The main contributions of the work proposed in this chapter can be summarized as follows:

- Provides an abstract encoding mechanism for positive expressions. It allows variable length in the condition part of a fuzzy rule.
- Develops a fuzzy set tuning scheme for the encoding mechanism proposed above.
- Defines a family of Michigan approaches for evolving fuzzy rules. This family is determined by the iterative process and the fuzzy class binarization used.
- Introduces a fitness function that takes into account the training data set, and length of the fuzzy rule.
- Describes completely a process (from the collecting data step up to the rule extraction mechanism) for genetic fuzzy rule based systems.

Experiments conducted showed that the proposed approach works well in detecting different attacks. The accuracy of fuzzy classifiers was good and comparable to those reported in the literature. The evolved fuzzy rules are not complex as no more than five attributes are used in each rule. Simpler fuzzy rules have a clear advantage in real applications. First, they yield rules that are easier to interpret. Second, they yield a classifier rule that is faster in deployment. This is especially crucial for data involving a large number of attributes. The importance of fast response is even more crucial in network security applications. In automatic intrusion detection systems, a fast classifier can mean all the difference between being able to detect and stop the
intruders behavior in time, as compared to detecting the users behavior, but responding too late because of a slow detection process. It is important to remember that training is the most time consuming part of this system.

However, the approach proposed in this chapter (EFR) requires information from both classes normal and abnormal when applied to the intrusion detection problem. If only information about the normal space is known, this approach cannot be used. The next chapter develops a technique that can be used in such kinds of scenarios.
Chapter 5

Fuzzy Self/Non-self Discrimination

5.1 Introduction

As shown in chapter 2, approaches based on artificial immune systems (AIS) have been applied successfully to perform anomaly detection [30, 86, 81, 99]. One major difference with other anomaly detection techniques is that AIS builds a model of the abnormal instead of the normal. This abnormal model is described in terms of a set of anomaly detectors which are generated by an algorithm called negative selection [50]. Depending on the representation used, it is possible to see these detectors as signatures of unknown attacks.

Dasgupta and Gonzalez proposed an anomaly detection technique, which uses the NS algorithm, for evolving detectors in the non-self (abnormal) space using a genetic algorithms in [33]. The evolved detectors possess a hyper-rectangular shape that can be interpreted as rules. The main contribution of that work was to use a real representation instead of binary strings.

This chapter is devoted to extending the idea of Dasgupta and Gonzalez by evolving fuzzy rule detectors instead of hyper-rectangles. Basically, a detector is a fuzzy rule that discriminates the abnormal behavior (non-self) from the normal behavior (self). A notion of volume and distance between fuzzy rules are introduced in order to apply AIS concepts. Moreover, this chapter builds a bridge between the framework for evolving fuzzy rules proposed in chapter 4 and Fuzzy Self/Non-self discrimination.
5.2 First Approach to Fuzzy Self/Non-self Discrimination (FD)

In [71], a fuzzy representation of rule detectors to produce a fuzzy measure of deviation from the self (normal) was proposed. A genetic algorithm with deterministic crowding (DC) was used for evolving such fuzzy rule detectors. A fuzzy rule detector was defined as:

\[
\text{If } x_1 \in T_1 \land \ldots x_n \in T_n \text{ then non_self,}
\]

where, \( x = (x_1, \ldots, x_n) \) is an element of the self/non-self space being evaluated; \( T_i \) is the fuzzy set defined by the union of some fuzzy sets in the fuzzy space\(^1\) of the attribute \( i \) (a restricted-sum-or fuzzy logic operator was used); \( \land \) is a fuzzy conjunction operator (\textit{min-and} was applied).

5.2.1 Encoding

Each atomic condition, \( x_i \in T_i \), is represented with a sequence \( (s_{i1}, \ldots, s_{im}) \) of bits, where \( m = |S| \) (the size of the fuzzy space for the attributes set of linguistic values)\(^2\). The bit \( s_{ij} \) is ‘on’ if and only if the corresponding basic fuzzy set \( S_j \) is part of the composite fuzzy set \( T_i \).

Figure 5.1 shows the structure of a chromosome.

![Figure 5.1: Structure of the chromosome representing the condition part of a rule.](image)

5.2.2 Fitness Function

The fitness of a fuzzy rule detector \( R \) is calculated using equations 5.1:

\[
\text{fitness}(R) = C \cdot (1 - \text{covering}(R)) + (1 - C) \cdot \text{volume}(R)
\]

\[
\text{covering}(R) = \frac{\sum_{x \in \text{Self}} \text{eval}_R(x)}{|\text{Self}|} 
\]

\[
\text{volume}(R) = \prod_{i=1}^{n} \text{measure}(T_i) 
\]

Here, \( \text{covering}(R) \) calculates the number of self samples being matched by the fuzzy rule detectors, \( \text{measure}(T_i) \) is the area under the membership function of the fuzzy set \( T_i \), and \( C \) is a penalty coefficient for covering self samples \((0 \leq C \leq 1)\). The higher the coefficient, the higher the penalty.

\(^1\)A fuzzy space is a collection of fuzzy sets (linguistic values) that are defined for an attribute, see chapter 2.

\(^2\)In this case, all the attributes have associated the same fuzzy space.
5.2.3 Similarity and Distance Definition

A notion of distance between individuals is required for replacing individuals and maintaining niches in DC [117]. The Hamming distance was used since each bit in the chromosome represents a unique fuzzy set [71].

5.2.4 Self/Non-self Discrimination

After the fuzzy rule detectors are generated, they are used for determining the abnormality (non/self) level of an element. An element $x$ is classified as non-self (abnormal) with a degree of abnormality equal to the highest truth value of the fuzzy rule detectors evaluated on such an element. A value close to zero means that $x$ is normal and a value close to one indicates that it is abnormal. Although this approach was successfully applied to several anomaly detection problems, there are several elements that can be improved. First, the fuzzy rule detectors evolved with this approach are not as simple as expected since they are defined in terms of all the attributes. Second, the fuzzy space is fixed and equal for each attribute; If such fuzzy space is not well designed, the fuzzy-immune approach may not generate a good set of fuzzy rule detectors. Finally, this approach is time consuming because several generations are required for evolving a good set of fuzzy rule detectors (in [71], the number of iterations was close to 1000 for the tested data sets).

5.2.5 Approach Disadvantages

Although applied with relative success in solving some intrusion detection problems [71], this approach has some disadvantages, including the following:

1. The encoding scheme does not allow one to generate simple fuzzy rules or modify the shape of the fuzzy sets.

2. The training is time consuming since it requires a large number of generations to generate a good set of candidate detectors.

3. The number and shape of the fuzzy sets associated with each attribute should be given in advance.

4. The search space grows exponentially in the number of attributes. If the dimension of the data set is $n$ and $k$ fuzzy sets are used for each attribute then the size of the chromosome is $n \times k$, i.e., the size of the search space is $2^{n \times k}$. 
5.3 Structured Fuzzy Self/Non-self Discrimination (S-FD)

In order to remove disadvantages 1 and 2 of FD, i.e., for generating simple fuzzy rule detectors and speeding up the training process, it is possible to extend the fuzzy rule detector definition as follows [72]:

\[
\text{If } x_{i_1} \in T_{i_1} \land \ldots \land x_{i_p} \in T_{i_p} \text{ then non\_self}
\]

Here, \( x = (x_1, \ldots, x_n) \) is an element of the self/non-self space being evaluated, \( \{i_k\}_{k=1,\ldots,p} \subseteq \{1,2,\ldots,n\} \) is a set of indices, \( T_{i_k} \) is fuzzy set defined by the union of some fuzzy sets in the fuzzy space of the attribute \( i_k \) (a restricted-sum-or fuzzy logic operator was used), and \( \land \) is a fuzzy conjunction operator (\text{min-and}). This definition is more general than the FD one and allows the definition of fuzzy rule detectors in terms of a subset of attributes.

5.3.1 Encoding

For encoding the fuzzy rule detectors in a chromosome, a structured representation proposed in [35] can be used. The Structured Genetic Algorithm (StGA) utilizes a multi-layered structure that provides a gene activation mechanism and allows the codification of redundant genetic material. A two-level StGA representation for encoding the fuzzy rule detectors is enough for representing the fuzzy rule detectors; the high level bits correspond to the set of indexes (features used) while the low level will represent the condition part of the fuzzy rule detector in the same way that was used in FD. The length of the high level portion of the chromosome is the same as the dimension of the feature space, \( n \). Each bit in the high level portion of the chromosome indicates whether a feature is used in the condition or not. Figure 5.2 shows the structure of the chromosome which is \( n + n \times m \) bits long (\( n \) is the dimension of the space and \( m \) is the number of basic fuzzy sets).

\[
\begin{align*}
\text{level 1} & \quad \text{level 2} \\
\left( x_1 s_1^1 \ldots s_m^1 x_2 s_1^2 \ldots s_m^2 \ldots x_n s_1^n \ldots s_m^n \right) \\
\text{gene 1} & \quad \text{gene n}
\end{align*}
\]

Figure 5.2: Structure of the chromosome representing a structured fuzzy rule detector.
5.3.2 Fitness Function

The fitness function of the S-FD is the same as the fitness function on FD but it takes into account only the attributes that have the associated high level bit 'on' (set of indexes).

5.3.3 Distance Definition

Two steps should be executed in order to calculate the distance between two rules represented by two level structured individuals. First, replace the set of low level bits associated with an 'off' high level with 1s. From a semantical point of view, a fuzzy rule that does not include an attribute \((x)\) in its condition \((C)\), is equivalent to one with condition \(C \land x \text{ is } T\) such that \(T\) equal to the union of all the fuzzy sets associated with such an attribute. Second, apply a Hamming distance between the transformed low level bits of both individuals.

5.4 Fuzzy Self/Non-self Discrimination with Variable Length Encoding and Fuzzy Sets Tuning (T-FD)

In order to “virtually” reduce the search space size and allow the evolutionary algorithm to tune the fuzzy sets associated with each attribute, it is possible to use the encoding mechanism of fuzzy rules proposed in chapter 4; one would use a variable-length chromosome with fuzzy set tuning. The idea is to define a fuzzy rule detector as follows:

\[
\text{If } x_{i_1} \in T_{i_1} \land \ldots x_{i_p} \in T_{i_p} \text{ then non_self}
\]

Here, \(x = (x_1, \ldots, x_n)\) is an element of the self/non-self space being evaluated, \(\{i_k\}_{k=1,\ldots,p} \subseteq \{1, 2, \ldots, n\}\) is a set of indexes, \(T_{i_k}\) is fuzzy set defined by a set of parameters, and \(\land\) is a fuzzy conjunction operator \((\min\text{-and})\). This definition is more general than the S-FD one and allows the tuning of fuzzy sets.

This definition can be seen as a particular case of the Extended Conjunctive Normal Form (XCNF-1) introduced in chapter 4, but restricting each atomic expression to have the form \(x \text{ is } T\) (negation is not allowed at the atomic level). Therefore, it is possible to use the same encoding strategy and genetic operators to evolve such fuzzy rule detectors.
Algorithm 12 Rule Distance

RULEDISTANCE(\(R_1, R_2, n\))
1. \(\text{for } i = 1 \text{ to } n \text{ do}\)
   2. \(\text{set}_1[i] = \text{trapezoid}(0, 0, 1, 1)\) // sets by default the biggest possible trapezoidal set
   3. \(\text{set}_2[i] = \text{trapezoid}(0, 0, 1, 1)\)
   4. \(l_1 = \text{length}(R_1)\) // determines the length of the first rule
   5. \(\text{for } i = 1 \text{ to } l_1 \text{ do}\)
      6. \(c = \text{atomic}(i, R_1)\) // gets the i-th atomic expression in rule R1
      7. \(\text{set}_1[\text{varId}(c)] = \text{fuzzyset}(c)\) // sets the fuzzy set associated with the attribute
      8. \(l_2 = \text{length}(R_2)\) // determines the length of the second rule
   9. \(\text{for } i = 1 \text{ to } l_2 \text{ do}\)
      10. \(c = \text{atomic}(i, R_2)\) // gets the i-th atomic expression in rule R2
      11. \(\text{set}_2[\text{varId}(c)] = \text{fuzzyset}(c)\) // sets the fuzzy set associated with the attribute
      12. \(\text{return } \sum_{i=1}^{n} \text{distance}(\text{set}_1[i], \text{set}_2[i]) / n \) // average distance between fuzzy sets

5.4.1 Distance Definition

Although DC requires a notion of distance between individuals, it is possible to extend DC in order to use a quasi-metric instead of a distance. A quasi-metric has the same properties of a distance but without requiring the symmetry property. Quasi-metrics are useful when comparing sets and rules. Given two sets \(s_1\) and \(s_2\), the distance of \(s_1\) to \(s_2\) \((d(s_1, s_2))\) is defined as the proportion of elements of \(s_2\) that are not in \(s_1\). In order to extend this definition to fuzzy sets, this quasi-metric can be seen as the area under the membership function of \(s_2\) that is not contained in the area under membership function of \(s_1\).

Notice that any fuzzy rule with a variable length condition can be extended in such a way that includes an atomic condition for each attribute: If the condition of a fuzzy rule \(R_1\) is defined in terms of \(k\) atomic expressions, \(a_1 \text{ AND } ... \text{ AND } a_k\), none being defined in terms of attribute \(i\), then such a condition is equivalent to the condition \(a_1 \text{ AND } ... \text{ AND } a_m \text{ AND } x_i\) is \(\text{trapezoid}(0, 0, 1, 1)\). Using these property, the distance from a fuzzy rule \(R_1\) to a fuzzy rule \(R_2\) can be defined as the average distance between the fuzzy sets of its attributes to the fuzzy sets of the attributes of rule \(R_2\), see Algorithm 12.

It is possible to prove that the distance from fuzzy rule \(R_1\) to fuzzy rule \(R_2\) is 0 if the degree of abnormally predicted by \(R_1\), for any sample \(x\), is higher than the degree of abnormally predicted by \(R_2\). In other words, anything that \(R_2\) predicts as abnormal is predicted by \(R_1\) as abnormal too. It is said that \(R_1\) dominates \(R_2\) and \(R_2\) is dominated by \(R_1\) when \(d(R_1, R_2) = 0\).
5.4.2 Fitness Function

The fitness function on T-FD is the same fitness function used for evolving fuzzy rules, (chapter 4, equation 4.10). Since no samples of the positive class are given, equation 4.10 reduces to equation 5.2.

\[
\text{fitness}(R) = w \ast \frac{\text{NN}(R)}{\text{NN}(R) + \text{NP}(R)} + (1 - w) \ast \left(1 - \frac{\text{FRL}(R)}{M}\right) \tag{5.2}
\]

Here, \(w\) is the weight associated with the fuzzy accuracy reached by the individual and \(M\) is the maximum number of atomic expressions defining a fuzzy rule. Although it looks like there is no correspondence between these fitness function and the one used for FD and S-FD, it is possible to show that both fitness function definitions are the same but calculating the “volume” of a fuzzy rule detector in a different way.

Using the notation introduced in chapter 4 and some fuzzy logic properties, the \(\text{covering}(R)\) function defined by 5.1 can be rewritten as follows:

\[
\text{covering}(R) = \sum_{x \in \text{Self}} \frac{\text{eval}_R(x)}{|\text{Self}|} \left| \text{Self} - \sum_{x \in \text{Self}} (1 - \text{eval}_R(x)) \right| = 1 - \frac{\sum_{x \in \text{Self}} \text{eval}_R(x)}{|\text{Self}|} = 1 - \frac{\text{NN}(R)}{\text{NN}(R) + \text{NP}(R)}
\]

Therefore, the fitness function used by FD and S-FD can be rewritten as follows:

\[
\text{fitness}(R) = w \ast \frac{\text{NN}(R)}{\text{NN}(R) + \text{NP}(R)} + (1 - w) \ast \text{volume}(R)
\]
\[
\text{volume}(R) = \prod_{i=1}^{n} \text{measure}(T_i) \tag{5.3}
\]

In the case of T-FD, the volume is approximated as \(\text{volume}(R) = \left(1 - \frac{\text{FRL}(R)}{M}\right)\). Clearly, T-FD requires much less time for determining the “volume” of a fuzzy rule detector than FD and S-FD.

5.4.3 Additional Considerations

Two additional issues must be tackled before being able to use T-FD: maintaining diversity, and allowing a fuzzy rule detector to cover additional non/self area with the same condition length.
5.4.3.1 Maintaining Diversity

Since the size of the search space is “virtually” reduced by introducing a variable length representation, the following condition is introduced in the deterministic crowding in order to reduce the crossover interaction problem, see chapter 3:

\[
\text{IF (fitness(child) > fitness(parent) AND (child dominates parent OR child is dominated by parent))}
\]

\[
\text{THEN replace(parent, child)}
\]

Such a condition allows the replacement of a fuzzy rule detector that dominates or is dominated by an offspring with higher fitness.

5.4.3.2 Increasing a Fuzzy Rule Detector Volume

Suppose that in a given generation of T-FD, a genetic operator, like mutation, produces an offspring fuzzy rule \( R_2 \) from a fuzzy rule \( R_1 \) such that \( R_2 \) dominates \( R_1 \), the length of \( R_1 \) is equal to the length of \( R_2 \) and the fuzzy accuracy of both of them is the same. Then, the fitness of \( R_1 \) is equal to the fitness of \( R_2 \) and \( R_1 \) will not be replaced by \( R_2 \) regardless that \( R_2 \) covers more non/self space than \( R_1 \). In order to allow the replacement of individuals with the same fitness that are dominated by other, the following replacement condition is introduced in the DC:

\[
\text{IF (fitness(child) = fitness(parent) AND child dominates parent THEN replace(parent, child)}
\]

5.4.4 Extracting the Final Set of Fuzzy Rule Detectors

Fuzzy rule detectors evolved by the EA (FD, S-FD or T-FD) are sorted by fitness from highest to lowest. Detectors with a fitness lower than the population average are removed. The average fitness is a good indicative of the “performance” of the detectors. Then, detectors that are dominated by other detectors with a higher fitness are removed because they are not covering additional non-self space and can cover self space. Finally, detectors that are dominated, regardless their fitness, are removed.

5.5 Experimentation with Machine Learning Data

Experiments were conducted on the machine learning two-class classification problems defined in section 2.8: Breast, Heart and Pima.
5.5.1 Experimental Settings

The data set is divided into 10 random groups of similar size. Then, each group is used as testing set of the fuzzy rule evolved by FD, S-FD, and T-FD using the normal samples of the remaining 9 groups as a training set. This validation techniques can be seen as a 10 cross validation without using the abnormal samples for training purposes, but only for testing. This procedure is repeated 5 times and the reported results are the average of such 50 runs. Each evolutionary algorithm is run for a maximum of 30 generations with a population size of 100 individuals. The accuracy weight is set to $w = 0.9$ (in order to give more importance to the correct discrimination of the normal samples than to the area covered by the fuzzy rule detector), the Min-And fuzzy operator is used, single point crossover and single bit mutation are used in FD and S-FD while single bit mutation (SBM), variable length crossover (VLSPX), gene addition (ADD) and extended gene deletion (XDEL) are used in the T-FD approach. Also, FD and S-FD associated 5 fuzzy sets with each attribute while T-FD used 6 divisions (in order to perform a fair comparison against FD and S-FD). Finally, each chromosome in the T-FD approach is initialized with a random length of 1 to $n$ genes where $n$ is the number of different attributes.

5.5.2 Results and Analysis

5.5.2.1 Comparison between T-FD, S-FD, and FD

Table 5.1 presents the average accuracy reached by T-FD, S-FD and FD, along with the average number of fuzzy rule detectors generated by each approach. A value of $a \pm b$ indicates that the average values was $a$ with standard deviation of $b$.

<table>
<thead>
<tr>
<th></th>
<th>T-FD</th>
<th>S-FD</th>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy %</td>
<td>Detectors</td>
<td>Accuracy %</td>
</tr>
<tr>
<td>BREAST</td>
<td>92.85±3.61</td>
<td>31.78±5.27</td>
<td>92.74±4.10</td>
</tr>
<tr>
<td>PIMA</td>
<td>65.47±4.66</td>
<td>42.38±4.19</td>
<td>66.75±4.58</td>
</tr>
<tr>
<td>HEART</td>
<td>62.44±9.04</td>
<td>38.10±4.60</td>
<td>61.11±7.99</td>
</tr>
</tbody>
</table>

As expected, the performance of FD is low compared to the performance of T-FD and S-FD since it is not able to explore appropriately the search space in few generations. Thus, FD generates a high number of detectors compared to the number of detectors generated by S-FD and T-FD. Notice that the T-FD performs similarly to S-FD since the dimensionality of these data sets is low (less than 14 dimensions).
5.5.2.2 Fuzzy Rule Complexity

Although the T-FD and S-FD performance is almost the same, T-FD generates more fuzzy rule detectors than S-FD. However, the complexity of T-FD generated detectors is low compared to the detectors generated by S-FD. While a fuzzy rule detector in S-FD can represent a non-connected region, a T-FD fuzzy rule detector always represents a connected region. Here, T-FD scores in interpretability. In general, a fuzzy rule detector generated by S-FD is equivalent to many fuzzy rule detectors generated by T-FD.

Figure 5.3 shows the evolution of the fuzzy rule length on T-FD for the best individual and the average of the population. Clearly, T-FD produces simple fuzzy rules, as no more than 3 attributes are included in the condition part of the fuzzy rules.

The following is a fuzzy rule detector generated by T-FD for the HEART data set in a sample run:

\[
\text{IF } x_{12} \text{ is } set_{2,5} \text{ AND } x_3 \text{ is } set_{2,6} \text{ THEN Heart-Disease}
\]

Here, \(set_{x,y}\) represents the trapezoidal fuzzy \(\{0, \frac{x-1}{m}, \frac{x}{m}, \frac{y}{m}, min \{1, \frac{y+1}{m}\}\}\). As can be noticed, the evolved fuzzy rules are easily interpretable by a human being.

5.5.2.3 Comparison with Results Reported in the Literature

Rows 1 and 2 of Table 5.2 presents the average performance reached by T-FD and S-FD along with some results reported in the literature and the approach proposed in chapter 4 (third row)\(^3\).

As expected, EFR performs better than T-FD and S-FD since it has access to the abnormal information during the training process. This information allows EFR to produce a better characterization of the normal/abnormal boundary. Considering that T-FD and S-FD have only access to the normal information,

\(^3\)Results reported for QDA, LDA, C4.5, kNN, SSV and FSM taken from [159].
Table 5.2: Comparative performance of the proposed approach

<table>
<thead>
<tr>
<th>Method</th>
<th>BREAST</th>
<th>PIMA</th>
<th>HEART</th>
<th>Statistical Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-FD</td>
<td>92.85</td>
<td>65.47</td>
<td>62.44</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>S-FD</td>
<td>92.74</td>
<td>66.75</td>
<td>61.11</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>EFR</td>
<td>94.85</td>
<td>74.21</td>
<td>78.96</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>QDA</td>
<td>94.90</td>
<td>74.80</td>
<td>57.80</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>LDA</td>
<td>96.00</td>
<td>77.20</td>
<td>60.40</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>C4.5</td>
<td>94.70</td>
<td>73.00</td>
<td>22.90</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>kNN</td>
<td>96.90</td>
<td>71.90</td>
<td>65.60</td>
<td>Leave-one-out</td>
</tr>
<tr>
<td>SSV</td>
<td>96.30</td>
<td>73.70</td>
<td>-</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>FSM</td>
<td>96.90</td>
<td>-</td>
<td>-</td>
<td>10-cross-validation</td>
</tr>
<tr>
<td>WM</td>
<td>87.10</td>
<td>71.30</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GIL</td>
<td>90.10</td>
<td>73.10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ABD</td>
<td>96.00</td>
<td>75.90</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ABA</td>
<td>95.10</td>
<td>74.80</td>
<td>-</td>
<td>random (50-50)%</td>
</tr>
</tbody>
</table>

results for T-FD and S-FD can be considered good. Moreover, such results are similar to results reported in the literature using all of the normal and abnormal information.

5.6 Experimentation with Intrusion Detection Data Sets

5.6.1 Darpa 99

5.6.1.1 Experimental Settings

The score of the system of fuzzy rule detectors evolved using the training data set is calculated as the classification accuracy over the testing set. This process is repeated 10 times and the reported results are the average of these 10 runs. Each evolutionary algorithm is run for a maximum of 30 generations with a population size of 100 individuals. The accuracy weight is set to $w = 0.9$, Min-And fuzzy operator is used, single bit mutation (SBM), variable length crossover (VLSPX), gene addition (ADD) and extended gene deletion (XDEL) are used in the T-FD approach while single bit mutation and single point crossover are used in the FD and S-FD approaches. FD and S-FD associated 5 fuzzy sets to each attribute while T-FD used 6 divisions (in order to perform a fair comparison against FD and S-FD). The initial population of T-FD is generated randomly with chromosome lengths of 1 to 9 – the number of attributes.

---

*Although all of these results were obtained using different portions of the data set and with different statistical validation methods (leave-one-out, or 10-cross-validation) or without statistical validation, the values reported here are indicative of the performance of the proposed approach.*
5.6.1.2 Results and Analysis

Figure 5.4 compares the ROC curve generated by each of the tested approaches (T-FD, S-FD and FD). As can be seen, the difference between the performance reached by T-FD and S-FD is minimal.

![ROC curves generated by the Fuzzy discrimination approaches on the Darpa99 data set.](image)

Table 5.3 presents the average detection rate, false alarm rate reached by T-FD and S-FD along with the average number of fuzzy rule detectors generated when the false alarm rate is set to 2.0%.

<table>
<thead>
<tr>
<th></th>
<th>FA%</th>
<th>DR %</th>
<th>Detectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-FD</td>
<td>2.00</td>
<td>93.94±0.87</td>
<td>11.5±1.18</td>
</tr>
<tr>
<td>S-FD</td>
<td>2.00</td>
<td>93.94±2.13</td>
<td>9.8±2.94</td>
</tr>
</tbody>
</table>

As expected, the performance of T-FD and S-FD is very similar. However, these results indicate that T-FD has a more consistent behavior than S-FD. The variance using T-FD on both detection rate and number of fuzzy rule detectors generated is lower than using S-FD.

5.6.2 KDDCup99

5.6.2.1 Experimental settings

A collection of 2000 normal data records is drawn from the data set and used as training data set while the remaining part of the data set is used as testing data. The score of the trained classifier is calculated as the
classification accuracy over the testing set. This process is repeated 10 times and the reported results are the average of these 10 runs. The accuracy weight is set to \( w = 0.9 \), Min-And fuzzy operator are used, single bit mutation (SBM), variable length crossover (VLSPX), gene addition (ADD) and extended gene deletion (XDEL) gene are used in the T-FD approach while single bit mutation and single point crossover are applied in the S-FD approach. Also, 6 divisions are allowed for tuning a fuzzy set while 5 fuzzy sets are associated to each attribute in the S-FD approach. The performance of T-FD and S-FD is calculated after 30, 50, 80 and 100 generations of the evolutionary algorithm with a population size of 100 individuals. These experiments are conducted in order to determine the effect of number of iterations in the quality of the solution produced by the proposed approaches (T-FD and S-FD).

5.6.2.2 Results and Analysis

Figure shows the ROC curves generated by T-FD and S-FD when the maximum number of generations was set to 30, 50, 80 and 100.

Clearly, T-FD outperforms S-FD regardless of the number of generations. Since the variable length encoding mechanism of T-FD reduces the search space size, it is able to evolve fuzzy rule detectors that cover large regions of the abnormal space in a few generations while keeping the false alarm rate low. It is not the case of S-FD, since the search space of S-FD is huge. It produces fuzzy rule detectors that cover small regions of the abnormal space. Table 6.7 compares the detection rate reached by S-FD and T-FD when the false alarm rate is fixed to 5% along with the number of fuzzy rule detectors generated.

Table 5.4: Detection Rate of T-FD and S-FD in the KDD Cup 99 problem when the false alarm is set to 5%.

<table>
<thead>
<tr>
<th>Generations</th>
<th>T-FD</th>
<th>S-FD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DR %</td>
<td>Detectors</td>
</tr>
<tr>
<td>30</td>
<td>69.20±23.01</td>
<td>45.9±3.63</td>
</tr>
<tr>
<td>50</td>
<td>94.75±7.06</td>
<td>43.6±3.57</td>
</tr>
<tr>
<td>80</td>
<td>97.60±1.85</td>
<td>40.4±3.75</td>
</tr>
<tr>
<td>100</td>
<td>98.28±1.56</td>
<td>40.9±4.12</td>
</tr>
</tbody>
</table>

Figure 5.6 compares the ROC curves generated by T-FD after different number of generations. When the number of generations is increased, T-FD is able to increase the detection rate while keeping the false alarm rate low.

If each attack is considered independently, T-FD generates the ROC curves shown in Figure 5.7 after evolving fuzzy rule detectors for 100 generations.
Figure 5.5: ROC curves generated by T-FD and S-FD on the KDDCup 99 data set when they were allowed to evolve fuzzy rule detectors for a maximum of (a) 30, (b) 50, (c) 80 and (d) 100 generations.

Figure 5.6: ROC curves generated by T-FD on the KDDCup 99 data set after different number of generations.
As expected, the behavior of the ROC curve considering only DoS attack is almost the same as considering all the attacks. This is due to the fact that DoS includes more than 99% of the attacks.

The classification accuracy of T-FD, when the false alarm rate is set to 8.4%, is shown in table 6.8 along with the performance achieved by the evolution of fuzzy rule mechanism (EFR) proposed in chapter 4. A false alarm rate of 8.4 was selected in order to compare the detection rate of both methods at the same false alarm rate.

Table 5.5: Comparative performance of T-FD and EFR.

<table>
<thead>
<tr>
<th>Class</th>
<th>T-FD</th>
<th>EFR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>91.60%</td>
<td>91.60%</td>
</tr>
<tr>
<td>U2R</td>
<td>51.24%</td>
<td>55.77%</td>
</tr>
<tr>
<td>R2L</td>
<td>40.30%</td>
<td>66.45%</td>
</tr>
<tr>
<td>DOS</td>
<td>99.55%</td>
<td>96.52%</td>
</tr>
<tr>
<td>PRB</td>
<td>74.93%</td>
<td>89.45%</td>
</tr>
</tbody>
</table>

Notice that the detection rate of attacks different to DoS reached by T-FD is not so bad taking into account the proportion of samples belonging to those attacks. For example, T-FD is able to detect 74.93% of the Prb attacks. Considering the KDDCup problem as a two class problem, EFR performs better than T-FD. Figure 5.8 compares the ROC curves generated by both approaches. However, EFR is able to recognize more attacks of small classes (different from DoS) than T-FD. These results can be explained as the consequence of two elements. First, EFR only evolves a single fuzzy rule for detecting all the possible attacks. It can reduce the
accuracy of the EFR classification system. Second, EFR uses information of all the attacks, while T-FD do not use such information. Clearly, EFR has an advantage in knowing which regions of the non-self space are populated by samples of each attack. Since T-FD does not know where the attacks can be located, it will try to cover as much it can without paying attention to small regions where the small attacks are located. Such small attacks will be not detected.

![Figure 5.8: ROC Curves generated by T-FD and EFR.](image)

### 5.7 Summary

This chapter presented a new technique that allows to generate a set of fuzzy rules that characterize the non-self space (abnormal) using as input only self (normal) samples. This work introduces the basic idea of evolution of fuzzy rule detectors and extends it in two different ways using StGA, and using the mechanism of variable length and fuzzy set tuning proposed in chapter 4. These extensions are developed in order to eliminate the disadvantages of the original fuzzy self/non-self discrimination technique. Experimentsshowed that both extensions perform better than the original one. Also, the fuzzy rule discrimination with tuning of fuzzy sets reached similar performance to StGA when the dimensionality of the data set is low. However T-FD outperforms StGA when the dimensionality of the data set is high.

The following are the main advantages of the fuzzy Self/Non-self discrimination using variable length and fuzzy set tuning:
• It shows an improved accuracy in the anomaly detection problem. This can be attributed to the fuzzy representation of rules which reduces the search space, allowing the evolutionary algorithm to find better solutions.

• It generates a more compact representation of the non-self space by reducing the number of detectors. This is also a consequence of the expressiveness of the fuzzy rules and the variable length encoding mechanism.

• It speeds up the evolutionary process and improves the performance of the algorithm by using variable length representation for the fuzzy rules.

As can be noticed in the KddCup data set, this approach generates a false alarm rate higher than 4% in order to detect more than 95% of the attacks. This behavior is expected since several fuzzy rule detectors are evolved in order to cover the non-self space. However, this behavior is sometimes undesirable since a high false alarm rate can increase the cost of checking and validating the status of the monitored system. The next chapter introduces a technique based on clustering concepts for generating a model of normal behavior. Since the model is representing the normal behavior, it is expected to reduce the false alarm rate.
Chapter 6

Data Clustering using Randomized Interactions between Data Points

6.1 Introduction

In chapter 2, the gravitational clustering (GC) proposed by Wright in [161] was introduced. Since this algorithm has a cubic time complexity $O(n^3)$ in the size of the data set being clustered, it is not appropriated for clustering large data sets. Moreover, GC is not robust to noise and does not determine the number of clusters automatically.

In this chapter, a novel unsupervised robust clustering technique based on the Gravitational Law and the second Newton’s motion Law that runs in $O(n\sqrt{n})$ is proposed. This technique, called Randomized Gravitational Clustering (RGC), uses the same ideas of GC but redefines the clustering target (it is not doing hierarchical clustering) and the dynamics of the system. Instead of simulating the gravitational system in a precise manner, a randomized simulation is carried on. In this way, for moving a data point it does not require the information of any other point in the system but only the information of another single point randomly selected. A kind of cooling mechanism similar to the one in simulated annealing is introduced in order to eliminate the formation of a single cluster (big-crunch) and noisy information (robust clustering). This technique is analyzed in order to determine the initial gravitational automatically.

Also, two extension of RGC are developed. The first extension, called data clustering based on Randomized INteraction between data points (RAIN), uses different dynamic functions rather than Gravitational Law
for simulating the system. The second extension associates a fuzzy set to each cluster generated by RGC or RAIN.

Finally, the proposed clustering technique is tested with some machine learning and intrusion detection data sets.

6.2 Randomized Gravitational Clustering

6.2.1 Newton’s Laws of Motion

Kinematics is the science of describing and explaining the motion of objects in an universe by using some models ([43, 121, 150, 157]). Particularly, Newton’s Laws of Motion describe and explain with high precision the motion of macro-objects in our Universe. This section presents the Newton’s laws of motion for a moving object describing a straight line trajectory in the $n$-dimensional Euclidean space.

6.2.1.1 1-Dimensional Laws of Motion

Let $x$ be an object in the Euclidean space, and $t$ be a real number representing an instant in time. Let $x(t)$ be the object position at time $t$, $s(t)$ is the object speed at time $t$, $a(t)$ is the object acceleration at time $t$, and $F(t)$ is the force applied on the object at time $t$. The speed and acceleration of an object are defined respectively as:

$$s(t) = \frac{dx(t)}{dt}$$  \hspace{1cm} (6.1)
$$a(t) = \frac{ds(t)}{dt} = \frac{d^2x(t)}{dt^2}$$  \hspace{1cm} (6.2)

The speed of an object is the derivative function of the position function, and the acceleration is the derivative function of the speed function. If the speed or acceleration functions of an object is known, the position can be deduced as:\textsuperscript{1}:

$$s(t) = \int_0^t a(t) dt$$  \hspace{1cm} (6.3)

\textsuperscript{1}In general the acceleration function can be defined as a differential equation. Therefore, the speed and position functions are found by solving such a differential equation.
\[ x(t) = \int_0^t s(t)dt = \int_0^t \left[ \int_0^t a(t)dt \right] dt \]  

(6.4)

If the acceleration of the object is a constant function then, the speed and position are:

\[ s(t) = s(0) + at \]  

(6.5)

\[ x(t) = x(0) + s(0)t + \frac{a t^2}{2} \]  

(6.6)

where, \( x(0) \) is the initial object position, \( s(0) \) is the initial object speed, and \( a \) is the constant value for the object acceleration.

Equations (6.5) and (6.6) can be used to simulate the movement of an object when the position and speed functions cannot be determined analytically. In this way, the acceleration is considered constant during an interval of time \( \Delta(t) \), and the speed and position of the object at time \( t + \Delta(t) \) are approximated as:

\[ s(t + \Delta(t)) = s(t) + a(t) \Delta(t) \]  

(6.7)

\[ x(t + \Delta(t)) = x(t) + s(t) \Delta(t) + \frac{a(t) \Delta(t)^2}{2} \]  

(6.8)

Finally, if \( m_x \) is the mass of the object \( x \), then the force exerted on the object is defined according to Newton’s second motion Law as follows:

\[ F(t) = m_x a(t) \]  

(6.9)

6.2.1.2 \textit{n-Dimensional Laws of Motion for Straight Line Trajectories}

The \( n \)-dimensional laws of motion for straight line trajectories are the vectorial extension of the 1-dimensional laws of motion. Let \( x \) be an object in the \( n \)-dimensional euclidean space, that is moving in the direction given by the vector \( \vec{d} \), and \( t \) be a real number representing an instant of time. Let \( x(t) \) be the object position at time \( t \), \( v(t) \) be the object velocity at time \( t \), and \( a(t) \) be the object acceleration at time \( t \). The velocity and the acceleration vector are defined as the vectorial extension of (6.1) and (6.2) respectively:
\[ v(t) = \frac{s(t)}{\|\vec{d}\|} \vec{a} \tag{6.10} \]

\[ \vec{a}(t) = \frac{a(t)}{\|\vec{d}\|} \vec{d} \tag{6.11} \]

Where, \( \|\vec{d}\| \) is the magnitude of the direction vector \( \vec{d} \), and \( a(t) \) and \( s(t) \) are the acceleration and speed of the object in the moving direction \( \vec{d} \). If the acceleration is constant, then the position and velocity are given by the vectorial extension of (6.5) and (6.6) respectively:

\[ v(t) = v(0) + \vec{a} t \tag{6.12} \]

\[ x(t) = x(0) + v(0) \ast t + \frac{\vec{a} t^2}{2} \tag{6.13} \]

If the acceleration is not constant, then the movement of an object can be approximated by using the vectorial extensions of (6.7) and (6.8) respectively:

\[ v(t + \triangle(t)) = v(t) + \vec{a}(t) \triangle(t) \tag{6.14} \]

\[ x(t + \triangle(t)) = x(t) + v(t) \triangle(t) + \frac{\vec{a}(t) \triangle(t)^2}{2} \tag{6.15} \]

### 6.2.2 Gravitational Law

These days, the most accepted theory for explaining the structure of the Universe and the motion of celestial bodies is the gravitational law developed by Isaac Newton. The Universal Law of Gravitation states that:

“Each body exerts an attractive force on any other body. The force is directed along the direction joining the centers of the two bodies. Its intensity is directly proportional to the product of their masses, and inversely proportional to the square of their center’s distance.”

The force exerted from one object \( x \) over another object \( y \) is expressed by the following equation:

\[ F(t) = \frac{G m_x m_y}{d(x(t), y(t))^2} \tag{6.16} \]
where, $m_x$ and $m_y$ are the masses of the two objects, $d(x(t), y(t))$ is the Euclidean distance between the two objects, and $G$ is the Universal gravitational constant $(6.67 \times 10^{-11})$.

The gravitational force exerted by an object over other objects defines a gravitational field around the object, as shown in Figure 6.1.

![Figure 6.1: Gravitational field](image)

In this way, an object $y$ is exposed to the gravitational field of object $x$, that pulls $y$ toward the center of this field, with an intensity given by the gravitational force, and with the direction defined by the vector $\overrightarrow{d}(t) = x(t) - y(t)$. Using the direction vector $\overrightarrow{d}$, the gravitational force equation can be rewritten as:

$$F(t) = \frac{G m_x m_y}{\|\overrightarrow{d}(t)\|^2} \quad (6.17)$$

Figure 6.2.a shows the movement directions of two objects ($y$ and $z$) that are exposed to the gravitational field of a third object ($x$). In Figure 6.2.a, only the gravitational field of $x$ is taking into account, i.e., the gravitational fields of $y$ and $z$ are not considered.

To calculate the final direction of an object’s motion, it is necessary to calculate the magnitude of the gravitational force that is exerted on the object for every other object in the universe. Figure 6.2.b shows the final direction of motion and force magnitude of $y$ according to the magnitude of the gravitational force exerted by $x$ and $z$. The size of an object indicates its mass.

From Newton’s second law of motion (6.9), the acceleration and acceleration vector of an object, $y$, due to the gravitational field of an object, $x$, are given by:

$$a(t) = \frac{G m_x}{\|\overrightarrow{d}(t)\|^2} \quad (6.18)$$
Because the acceleration function is a complex differential equation, to find the position function of a given object under the influence of one or more gravitational fields is not an easy task. Instead, the movement of an object is approximated by using the acceleration vector given by (6.19) in the movement equations (6.14) and (6.15). Therefore, the movement equations of an object $y$ under the influence of the gravitational field of an object $x$ are:

$$v(t + \triangle(t)) = v(t) + \overrightarrow{d} \frac{Gm_x}{\|\overrightarrow{d(t)}\|^3} \triangle(t)$$

$$y(t + \triangle(t)) = y(t) + v(t) \triangle(t) + \frac{\overrightarrow{d(t)} Gm_x \triangle(t)^2}{2 \|\overrightarrow{d(t)}\|^3}$$

6.3 Clustering

For a $k$-dimensional data set with $n$ data points, each data point is considered as an object in the $k$-dimensional space with mass equal to 1. Each point in the data set is moved according to a simplified version of Equation (6.21). The basic ideas behind applying the gravitational law are:

1. A data point in some cluster exerts a higher gravitational force on a data point in the same cluster than on a data point that is not in the cluster. Therefore, points in a cluster move in the direction of the center
of the cluster. In this way, the proposed technique will determine automatically the clusters in the data set.

2. If some point is a noise point, i.e., does not belong to any cluster, then the gravitational force exerted on it from other points is so small that the point is almost immobile. Therefore, noise points will not be assigned to any cluster.

In order to reduce the amount of memory and a time expended in moving a data point according to the gravitational field generated by another point \((y)\), equation (6.21) is simplified as follows:

\[
x(t + 1) = x(t) + \frac{\vec{G}}{\|\vec{d}\|^3}
\]

(6.22)

here, \(\vec{d} = \vec{y} - \vec{x}\), and the constant \(G\) includes the division by 2 defined in 6.21. In this way, the velocity at any time, \(v(t)\), of an object is considered as the zero vector and \(\triangle(t) = 1\).

Since the distance between points is reduced each iteration, all the points will be moved to a single position after a huge (possibly infinite) number of iterations, \((big\ crunch)\). Then, the gravitational clustering algorithm will define a single cluster.

In order to eliminate this limit effect, the gravitational constant \(G\) is reduced each iteration in a constant proportion (the decay term: \(\triangle(G)\)). Algorithm 13 shows the randomized gravitational clustering algorithm.

### Algorithm 13 Randomized Gravitational Clustering

\[
\text{RGC}(x, G, \triangle(G), M, \varepsilon)
\]

1. for \(i = 1\) to \(n\) do // each data point is a candidate cluster
2. MAKE(i)
3. for \(i = 1\) to \(M\) do // iterations performed by the algorithm
4. for \(j = 1\) to \(n\) do // moving each data point
5. \(k =\) random point index such that \(k \neq j\) // selecting another data point
6. MOVE(\(x_j, x_k\)) (see Eq (6.22))//Move both points
7. if \(\text{dist}(x_j, x_k) \leq \varepsilon\) then UNION(\(j, k\)) // merging cluster if possible
8. \(G = (1 - \triangle(G)) \times G\) // Cooling the system
9. for \(i = 1\) to \(n\) do // organizing the final clusters
10. FIND(i)
11. **return** disjoint-sets

Function MOVE (line 6), moves both points \(x_j\) and \(x_k\) using equation 6.22 taking into consideration that both points cannot move further than half of the distance between them. In each iteration, RGC creates a set of clusters by using an optimal disjoint set union-find structure\(^2\) and the distance between objects (after

---

\(^2\)A disjoint set union-find structure is a structure that supports the following three operators [28]:

- **MAKESET(x)**: Create a new set containing the single element \(x\)

93
moving data points according to the gravitational force). When two points are merged, both of them are kept in the system while the associated set structure is modified. In order to determine the new position of each data point, the proposed algorithm merely selects another data point in a random way and moves both of them according to equation 6.22 (MOVE function). RGC returns the sets stored in the disjoint set union-find structure.

Because RGC assigns every point in the data set (noisy or normal) to one cluster, it is necessary to extract the valid clusters. We used an extra parameter ($\alpha$) to determine the minimum number of points (percentage of the training data set) that a cluster should include in order to be considered a valid cluster. In this way, we used an additional function GETCLUSTERS that takes the disjoint sets generated by RGC and returns the collection of clusters that have at least the minimum number of points defined, see Algorithm 14.

**Algorithm 14 Cluster Extraction.**

```
GETCLUSTERS( clusters, $\alpha$, n )
1 newClusters = $\emptyset$
2 MIN_POINTS = $\alpha n$
3 for i=0 to number of clusters do
4    if size( cluster$_i$ ) $\geq$ MIN_POINTS then
5        newClusters = newClusters $\cup$ { cluster$_i$ }
6 return newClusters
```

### 6.3.1 Time Complexity Analysis

The time complexity of the MOVE function is constant (the space dimension is considered constant in this analysis). The function UNION is executed in the inner for loop $j$ (lines 4-7) at most once, then the complexity of the inner loop is bounded by $n$ times the execution of the function UNION. Therefore the time complexity in practice of the inner loop $j$ is $O(n^3)$. The same analysis is done for loop in lines 9-10. The time complexity of the first loop (lines 1-2) is $O(n)$ because the time complexity of the MAKE function is constant. The time complexity to get the disjoint sets is $O(n)$ by traversing the array of canonical index and separating the objects according to it. The parameter $M$ defines the number of iterations that the algorithm will be executed. Therefore the time complexity of RGC is $O(Mn)$. Although it looks like RGC has linear time complexity, experimental results indicate that the number of iterations required by RGC to find a good set of clusters is

- **UNION($x$, $y$):** Replace the two sets containing $x$ and $y$ by their union.
- **FIND($x$):** Return the name of the set containing the element $x$.

In the optimal disjoint set union-find structure, each set is represented by a tree where the root of the tree is the canonical element of the set, and each child node has a pointer to the parent node (the root node points to itself) [28].

It has been proved that the time complexity of applying any sequence of $m$ UNION and FIND operations on $n$ elements is at most $O((m + n) \log^* n)$, where $\log^* m$ is the inverse function of the Ackerman function [28]. If $n \leq 2^{2^{2^{2^{2^{22}}}}} \approx 10^{19728}$ then $\log^* n \leq 5$. Therefore the time complexity of any sequence of $m$ UNION and FIND operations on $n$ elements is at most $O(m + n)$ in practice.
close to $10\sqrt{n}$. Therefore, upon experimental evidence, the time complexity of RAIN is $O(n\sqrt{n})$. It is clear that the time complexity of the GETCLUSTERS function is generally such that $O(\#\text{clusters}) \ll O(n)$. Then the time complexity of the complete algorithm (RGC and GETCLUSTERS) is $O(n\sqrt{n})$ in practice.

### 6.3.2 Comparison with Wright’s work

Although the proposed algorithm uses the gravitational law in the same way as Wright’s work [161], several differences can be established between them. Table 6.1 summarizes the main differences between them.

<table>
<thead>
<tr>
<th>Property</th>
<th>RGC</th>
<th>Wright’s [161]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>$O(n\sqrt{n})$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>End Condition</td>
<td>Number of iterations</td>
<td>Only one particle remaining</td>
</tr>
<tr>
<td>Particles Number</td>
<td>The same each iteration: $n_1$</td>
<td>Variable: according to the merging process</td>
</tr>
<tr>
<td>Particles mass</td>
<td>Constant = 1</td>
<td>Variable: according to the merging process</td>
</tr>
<tr>
<td>Gravity force</td>
<td>The same for each particle, but decreases with each iteration</td>
<td>Different for each particle, depends on the mass of the particle</td>
</tr>
</tbody>
</table>

### 6.3.3 Experimentation

Tests are conducted over three different synthetic noisy data sets (Gaussian clusters: elliptic and spherical), shown in figure 6.3.

![Figure 6.3: Synthetic data sets](image)

(a) Five clusters data set  (b) Three spherical clusters data set  (c) Three clusters elliptical data set
6.3.1 Experimental Settings

Each data set is normalized according to the maximum and minimum values found per attribute. The total number of points, \( N \), the number of noisy points, \( N_{n} \), and the generating center coordinates for these Gaussian clusters are listed in table 6.2, respectively. RGC is run using the following parameter values: number of iterations \( M = 500 \), initial gravitational force \( G = 7 \times 10^{-6} \), gravitational force decay \( \Delta(G) = 0.01 \), and minimum distance \( \varepsilon = 10^{-4} \).

6.3.2 Results

Table 6.2 shows the average and standard deviation on the performance over 10 different runs of RGC, k-means and fuzzy k-means. The number of iterations in the fuzzy k-means and the k-means algorithms were 150 in order to perform a fair comparison between the time expended by the RGC algorithm and these methods. It is clear that in the first data set (with five clusters), 100 iterations in k-means or fuzzy k-means expend the same or more time than 500 iterations of the RGC algorithm.

<table>
<thead>
<tr>
<th>Set</th>
<th>N</th>
<th>( N_{n} )</th>
<th>Generating centers</th>
<th>RGC</th>
<th>k-Means</th>
<th>fuzzy k-Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2530</td>
<td>626</td>
<td>(88,86)</td>
<td>(457 ± 6.4, 464 ± 1.46)</td>
<td>(58.0 ± 1.0, 52.3 ± 0.32)</td>
<td>(51.0 ± 0.16, 52.3 ± 0.04)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(151,100)</td>
<td>(152 ± 1.59, 80.5 ± 1.35)</td>
<td>(181 ± 0.53, 80.6 ± 0.32)</td>
<td>(187 ± 0.19, 185 ± 0.27)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(100,150)</td>
<td>(166 ± 27.7, 145 ± 1.93)</td>
<td>(81.7 ± 8.19, 165 ± 13.27)</td>
<td>(165 ± 24.15, 156 ± 13.76)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(170,160)</td>
<td>(170 ± 0.47, 161 ± 0.48)</td>
<td>(190 ± 0.70, 197 ± 0.76)</td>
<td>(195 ± 2.20, 193 ± 5.59)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(280,200)</td>
<td>(201 ± 0.70, 197 ± 0.76)</td>
<td>(195 ± 4.20, 193 ± 5.59)</td>
<td>(202 ± 7.92, 203 ± 4.69)</td>
</tr>
<tr>
<td>2</td>
<td>1157</td>
<td>346</td>
<td>(80,50)</td>
<td>(50.3 ± 1.29, 45.5 ± 1.86)</td>
<td>(65.8 ± 1.94, 67.3 ± 2.07)</td>
<td>(67.0 ± 0.04, 66.7 ± 0.04)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(90,90)</td>
<td>(89.5 ± 6.1, 89.1 ± 0.22)</td>
<td>(190 ± 17.06, 84 ± 11.18)</td>
<td>(113 ± 0.4, 169 ± 0.26)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(150,150)</td>
<td>(149 ± 9.4, 147 ± 0.41)</td>
<td>(124 ± 4.20, 168 ± 12.89)</td>
<td>(174 ± 3.00, 136 ± 0.33)</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>100</td>
<td>(50.50)</td>
<td>(50.4 ± 0.95, 49.2 ± 0.1)</td>
<td>(58.3 ± 9.46, 59.3 ± 9.46)</td>
<td>(82 ± 0.0, 49.6 ± 0.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(99.7 ± 9.45, 101.5 ± 2.25)</td>
<td>(152 ± 9.51, 80.4 ± 11.51)</td>
<td>(115 ± 0.0, 105 ± 0.0)</td>
<td>(115 ± 0.0, 105 ± 0.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(150,210)</td>
<td>(147 ± 6.3, 209 ± 7 ± 0.4)</td>
<td>(138 ± 2.42, 204 ± 2.45)</td>
<td>(148 ± 0.0, 207 ± 0.0)</td>
</tr>
</tbody>
</table>

As shown in table 6.2, RGC is robust to noise and in every case, the result were almost the same (standard deviation lower than 2.5 for all cluster centers), and close to the real centers. On the other hand, k-means and fuzzy k-means were very sensible to noise and the results varied from one run to the other drastically. In some case the standard deviation was very high (see third center in second data set). Figure 6.4 shows the clusters obtained by RGC over the data sets in a sample run. It is clear that the proposed approach is able to find the clusters independently of their shape.
Figure 6.4: Clusters obtained by the G-Algorithm in a sample run
In order to determine the sensitivity of RGC to the parameter values, each parameter was varied in a range of different values while keeping the other parameters constant.

### 6.3.3.3 Scalability and Dynamics of the G-Algorithm

Figure 6.5 shows the movement of the data points in a simple run after different number of iterations. Clearly, points inside clusters move in the direction of their cluster centers while noisy points stay in almost the same position, thus confirming the hypothesis of cluster agglomeration.

![Figure 6.5: Movement of data points in different iterations](image)

Figures 6.5b and 6.5c show the concentration of points around the cluster centers after different number of iterations. Due to the randomness in the RGC, some points inside clusters were not moved in the direction of the cluster centers. According to this behavior, it is possible that the RGC does not require the entire data set to determine the clusters.

In order to determine the scalability of the proposed approach, the RGC algorithm was run using different percentages of the data set (between 1% and 100% with differences of 1%). The portion of the data set used was randomly chosen from the original data set. This process was repeated 50 times and the reported results are the average over these repetitions (see Figure 6.6).

According to figure 6.6, RGC is able to find the clusters in the data set using only 20% (or more) of the data set. Therefore, it is not necessary to use the entire data set to obtain good results.

### 6.3.3.4 Sensitivity to $\alpha$

The RGC algorithm uses the parameter $\alpha$ to extract the valid clusters according to their size. For the experiments performed with the synthetic data sets, it was found that $\alpha = 0.03$ (clusters with size equal or bigger than 3% of the full data set) is a good value. Figure 6.7a shows the number of clusters for different values of $\alpha$. 
Figure 6.6: Number of clusters obtained by the G-Algorithm by sampling the data set.

\( \alpha \), while figure 6.7b shows the number of points covered by the clusters (points included in some cluster with the given size). These values are the average over 50 experiments that were performed.

Figure 6.7 suggests that heuristic techniques can be applied to determine an appropriate value of this parameter. Because this parameter is only used in the extraction of the “valid” clusters (does not affect the gravitational scheme), this parameter can be used for analyzing the internal structure of the data set after RGC is applied. For example, according to figure 6.7a the data set 1 has two big clusters, each one with a size of 10% of the data set size. Figure 6.8 shows the clusters obtained by varying the \( \alpha \) parameter in a sample run.

6.3.3.5 Sensitivity to the Initial Gravitational force (\( G \))

\( G \) is the most important parameter in the RGC. If it is fixed to a big value, RGC will form only one cluster (limit behavior). On the other hand, if it is fixed to a very low value, the RGC will not create clusters at all (see Figure 6.9). According to figure 6.9 (using logarithmic scale), the results of RGC rely heavily on the value given to this parameter.

After a careful analysis, it was not possible to define a “universal” value for this parameter, i.e., a value that works for all data sets (real or synthetic). Therefore, an optimal value for this parameter depends on the data set, and has to be chosen carefully.

6.3.3.6 Sensitivity to the Gravitational Force Decay (\( \Delta(G) \))

\( \Delta(G) \) is a very important parameter in the RGC algorithm. If it is fixed to a big value, for example, 0.1, RGC will decrease the gravitational force so fast, that the data points will not have a chance to form clusters at all.
Figure 6.7: Internal structure of the data set. Number of clusters and total points in the clusters.
Figure 6.8: Resolution levels in the G-Algorithm, as shown in Figure 6.7.

Figure 6.9: Sensitivity to initial gravitational force (log scale)
On the other hand, if it is fixed to a low value, for example, 0.001, RGC will decrease the gravitational force so slowly, that the data points will form only one cluster (the limit behavior), see Figure 6.10. According to figure 6.10, the RCG algorithm is very sensitive to this parameter. However, values between 0.015 and 0.001 showed good performance.

![Figure 6.10: Sensitivity to gravitational force decay](image)

6.3.3.7 Sensitivity to Merging Distance ($\varepsilon$)

The epsilon parameter, $\varepsilon$, was varied between 0.1 and $1e-6$ to determine the sensitivity of the RGC algorithm to this parameter. Figure 6.11 shows that the RGC is not very sensitive to this parameter. Only when this parameter is set to a big value ($\geq 0.01$), the results obtained by RGC become bad (only one cluster). This behavior is expected because a value of 0.01 indicates that two points at a distance of 0.1 are in the same cluster (0.1 is almost 10% of the maximum distance between any two points in the data set).

6.4 RAIN: Clustering based on RAndomized INteractions of data points

RAIN extends the RGC algorithm in such a way that different decreasing functions can be used (instead of the one based on the Gravitational Law) and removes the setting of the initial gravitational force. Basically, three elements are considered here: reducing the effect of the data set in the dynamics of the system, defining an interaction function, and setting automatically the initial interaction strength (gravitational force).
6.4.1 Maximum Distance between Closest Points

In order to reduce the sensitivity of RAIN to the size of the data set, we calculate a rough estimate of the maximum distance between closest points in the data set. This distance will allow RGC to have a reference value for merging and moving data points.

Given a collection of \( n \) data points in the \( k \)-dimensional \([0, 1]\) Euclidean space, the maximum distance between closest points can be roughly approximated by considering the data points arranged in a grid defining isosceles triangles (pyramids). Since the height of an isosceles triangle is \( \frac{2}{\sqrt{3}} \) the side of the triangle and the maximum number of points per dimension of such grid is bounded by \( n^{\frac{k}{2}} \) then such distance can be approximated using equation 6.23.

\[
\hat{d} = \frac{2 \cdot \sqrt{k}}{\sqrt{3} \cdot \sqrt{n}} \tag{6.23}
\]

Here, \( \sqrt{n} \) is a correction factor for data sets where the number of points is considerably low compared to the number of vertices on the hypercube \([0, 1]^k\), i.e., \((2^k)\).

6.4.2 Moving Functions

Although motivated by the Gravitational Law and Newton’s second law of motion, RGC can be seen as an algorithm that moves interacting data points according to a decreasing function of the data points’ distance. In RAIN, the final position of a data point \( x \) that is interacting with another data point \( y \) is defined by equation 6.21.
$$x(t + 1) = x(t) + G \cdot \vec{d} \star f \left( \frac{||\vec{d}||}{\hat{d}} \right)$$  \hspace{1cm} (6.24)

Here, $\vec{d} = \vec{y} - \vec{x}$, $f$ is a decreasing function, $\hat{d}$ is the rough estimate of maximum distance between closest points, and $G$ is the initial strength of the data points interaction. Although many decreasing functions can be used, in this work only $f(x) = \frac{1}{x^3}$ (the gravitational force function) and $f(x) = e^{-x^2}$ are considered.

### 6.4.3 Setting the Initial Interaction Strength ($G$)

Since RAIN creates the clusters while it moves the data points, it is possible to use the number of merged points after some checking iterations to determine if RAIN is using an appropriate value of $G$. A point is considered merged if it has been assigned to a cluster of size greater than one. A high number of merged points (more than a half) can indicate that the initial strength is so high that it will define a single big cluster. If the number of merged points is low (less than a half) it can indicate that the initial strength is so low that no cluster is going to be defined at all. Therefore, a value close to half of the number of data points will indicate an appropriate value for the initial strength. Algorithm 15 shows the process for determining the initial interaction strength.

**Algorithm 15** Initial Strength Estimation.

**GET INITIAL STRENGTH** ($x, \Delta(G), M, \varepsilon$) // M is set to $\sqrt{n}$

1. $G = 1$
2. RGC($x, G, \Delta(G), M, \varepsilon$) // test the given strength
3. $K =$ number of merged points
4. while $\frac{n}{2} - K > \sqrt{n}$ do
5. $G = 2 \times G$
6. RGC($x, G, \Delta(G), M, \varepsilon$) // test the given strength
7. $K =$ number of merged points
8. if $G > 1$ then $a = \frac{G}{2}$ else $a = 0$
9. $b = G$
10. $G = \frac{a + b}{2}$
11. RGC($x, G, \Delta(G), M, \varepsilon$) // test the given strength
12. $K =$ number of merged points
13. while $|\frac{n}{2} - K| > \sqrt{n}$ do
14. if $\frac{n}{2} > K$ then $b = G$ else $a = G$
15. $G = \frac{a + b}{2}$
16. RGC($x, G, \Delta(G), M, \varepsilon$) // test the given strength
17. return $G$

The full RAIN algorithm is shown in Algorithm 16.
Algorithm 16 RAIN.

RAIN(\(x, \Delta(G), M, \varepsilon, \alpha\)) \(\text{// M can be set between 1 and 10}\)
1 \(G = \text{GET INITIAL STRENGTH}(x, \Delta(G), \sqrt{n}, \varepsilon)\)
2 clusters = RGC(\(x, G, \Delta(G), M^*\sqrt{n}, \varepsilon\)) \(\text{// execute the RGC algorithm}\)
3 clusters = GETCLUSERS( clusters, \(\alpha, n\) )
4 return clusters

6.4.4 RAIN Time Complexity

Experimental results indicated that for checking each candidate initial strength, the number of iterations required is close to \(\sqrt{n}\). Therefore, upon experimental evidence the time complexity of RAIN is \(O(n\sqrt{n})\).

6.4.5 Experimentation

In order to evaluate the performance of RAIN, experiments are performed on four synthetic data sets; three of them are included in the CLUTO toolkit. Figure 6.12 shows the four synthetic data sets.

(a) Chameleon with 1000 points and 9 natural clusters
(b) Chameleon with 8000 points and 6 natural clusters
(c) Chameleon with 8000 points and 8 natural clusters
(d) Synthetic with 12000 points and 7 natural clusters

Figure 6.12: Synthetic data sets used with RAIN.
6.4.5.1 Experimental Settings

RAIN is run with a distance merging of $\varepsilon = 1e - 4$, decay term of $\Delta(G) = 0.001$, and minimum cluster size of $\alpha = 50$ points. RAIN is run with both of the defined interaction (moving) functions. The reported results are obtained after allowing RAIN to determine the initial strength and $5\sqrt{n}$ number of iterations ($n$ is the size of the data set).

6.4.5.2 Analysis of Results

Figure 6.13 presents the extracted clusters and configuration of data points after 100, 300, and 500 iterations of a typical run of RAIN for the chameleon 10000-9 data set using $e^{-x^2}$ as the interaction function.

![Figure 6.13: RAIN evolution for chameleon data set 10000-9 using $e^{-x^2}$ (best seen in color). Points assigned to the same cluster are drawn with the same color.](image)

Notice that after 100 iterations ($\sqrt{10000}$), almost all the points belonging to the natural clusters have been merged with other points in the cluster while not many noisy points have been merged, see Figure 6.13a.
Moreover, after 500 iterations, noisy points have been not practically moved from their original positions, see Figure 6.13f. Clearly, RAIN is able to deal with noise in the data set.

As the number of iterations is increased, the size of the conformed clusters increases too. After 300 iterations, RAIN is able to detect all the natural clusters, but has some of them divided in two or three clusters, see Figure 6.13b. When the maximum number of iterations is reached \(5\sqrt{10000}\), RAIN has detected all the natural clusters. Similar behavior was observed in several different trials.

Figure 6.14 shows the results obtained by RAIN in the other three data sets.

![Figure 6.14: RAIN clustering for synthetic data sets using \(e^{-x^2}\) (best seen in color).](image)

In general, RAIN using \(e^{-x^2}\) as interaction function is able to find the natural clusters in the data set. Sometimes RAIN merges two clusters (Figure 6.14b), and sometimes it generates two or more clusters instead of one (Figure 6.14a). Only for the chameleon 8000-8 data set, one cluster was missed. It is possible that being a cluster of such low density, RAIN is not able to conform it due to the random interaction process.

Figure 6.15 presents the extracted clusters and configuration of data points after 100, 300 and 500 iterations of a typical run of RAIN for the chameleon 10000-9 data set using \(\frac{1}{x^3}\) as interaction function.

As expected, this configuration of RAIN has similar behavior to the one previously discussed. However, some differences can be noticed in the dynamics of the system. When RAIN is using \(e^{-x^2}\), it creates some holes in the clusters, see Figure 6.13f, but when using \(\frac{1}{x^3}\) the clusters are well defined after 500 iterations, see figure 6.15f. Clearly, RAIN has detected all the natural clusters, but has divided two of them in two halves. Similar behavior was observed in several different trials.

Figure 6.16 shows the results obtained by RAIN in the other three data sets.
Figure 6.15: RAIN evolution for chameleon data set 10000-9 using $\frac{1}{x^3}$ (best seen in color).

Figure 6.16: RAIN clustering for synthetic data sets using $\frac{1}{x^3}$ (best seen in color).
In general, when RAIN is using $\frac{1}{x^3}$, it merges some close natural clusters. It is possible that this function requires some additional considerations for setting the initial interaction strength. However, the performance is not bad; no more than three clusters are merged together.

### 6.5 Classification using Fuzzy Cluster Prototypes

With each cluster generated by RAIN, it is possible to generate a fuzzy cluster prototype for the cluster by associating a fuzzy set membership function. Such a fuzzy set will determine the membership level of data points to the cluster. Fuzzy prototypes are used instead of crisp prototypes in order to eliminate the sharp distinction between the boundaries of the clusters generated with RAIN. The main goal of this process is to increase the accuracy in problems where the boundaries between clusters is not well defined.

#### 6.5.1 Cluster Statistical Information

In order to generate the fuzzy membership functions, the statistical information shown in table 6.3 is computed for each cluster $k = 1, \ldots, C$, where $C$ is the number of clusters generated by RAIN. This information is computed using only the data points that were assigned to the cluster.

Table 6.3: Cluster information generated after the gravitational clustering algorithm is applied. $S_k$ is the set of points that are assigned to the cluster $k$.

<table>
<thead>
<tr>
<th>Information</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>$center_k = \frac{1}{</td>
</tr>
<tr>
<td>Radius</td>
<td>$radius_k = \max_{1 \leq i \leq</td>
</tr>
<tr>
<td>Avg. Radius</td>
<td>$\text{radius} = \frac{1}{</td>
</tr>
<tr>
<td>Min values</td>
<td>$\min_{k,j} = \min_{1 \leq i \leq</td>
</tr>
<tr>
<td>Max values</td>
<td>$\max_{k,j} = \max_{1 \leq i \leq</td>
</tr>
</tbody>
</table>

#### 6.5.2 Fuzzy Cluster Membership

A fuzzy membership for each cluster is generated according to the statistical information shown in table 6.3. The membership of a data point to a cluster is determined by the combination of two fuzzy sets. One
is defined in terms of the center, average radius and the radius (hyper-sphere model). The other is defined in terms of the center, min values and max values (hyper-rectangle model). The hyper-sphere fuzzy set, is defined as shown in figure 6.17.

![Hyper-sphere fuzzy cluster membership function.](image)

The hyper-sphere fuzzy set is defined in four zones:

1. The first zone covers data points that have a distance to the cluster center that is lower than the average distance of points in the cluster. These points receive a high membership value (between 1.0 and 0.8). This zone can be seen as the **truth cluster** points region.

2. The second zone covers the data points with a distance higher than the average distance of the points in the cluster to the cluster center, but lower than the maximum distance of any point in the cluster to the cluster center. These points receive a membership between 0.8 and 0.5. This zone can be seen as **almost in cluster** points region.

3. The third zone covers points that are farther to the center than any point in the cluster, but to a maximum distance of two cluster radius. These points receive a membership value between 0.5 and 0.0. This zone can be seen as **almost non-cluster** points region.

4. The last zone covers points that are farther than twice the distance of any point in the cluster to the cluster center. These points receive a membership of 0.0 and define the **truth non-cluster** points region.

It is clear that any point that is in the cluster will receive a membership value to the cluster of at least 0.5. On the other hand, the hyper-rectangle fuzzy set is defined as:
where, \( g_{k,i} \) is a triangular fuzzy set defined as shown in figure 6.18. It is clear that a data point that belongs to the hyper-rectangle \([\text{min}, \text{max}]\) will receive at least a membership value of 0.5. Therefore, all the data points that are assigned to the cluster will receive at least a membership value of 0.5.

Finally, the fuzzy membership function associated with a given cluster is calculated as the intersection of the hyper-sphere fuzzy set with the hyper-rectangle fuzzy set. We used the average multiplication operator as intersection operator. Therefore, the combined membership function to the cluster \( k \) is given by:

\[
F_k(x) = \frac{2 * f_k(x) * g_k(x)}{f_k(x) + g_k(x)}
\]  

(6.26)

6.5.3 Prediction

Given the set of clusters generated by the gravitational clustering algorithm and the fuzzy membership associated with each cluster, a data point is classified in the cluster with the highest membership function value, see equation 6.27.

\[
x \in k \iff F_k(x) = \max \{ F_i(x) \} \forall i = 1, 2, ..., C
\]  

(6.27)

In the case that a point cannot be assigned to any cluster, i.e. the membership value for each cluster is zero, then the point is assigned to the cluster with the closest cluster center to the point, see equation 6.28:
An appropriated label can be given to each cluster following the strategy introduced in section 2.6.6.

### 6.5.4 Reviewing the Cluster Prototypes

Since RAIN might not assign every training data point to a cluster, it is possible to use the prediction mechanism in order to assign each training data point to some cluster. Then, the clusters prototype information can be recomputed in order to reflect such changes.

### 6.5.5 Experimentation

In order to determine the performance of RAIN in the (anomaly) classification problem, experiments were conducted on the machine learning two-class classification problems defined in section 2.8: Breast, Heart and Pima.

#### 6.5.5.1 Experimental Settings

A 10 fold cross-validation technique is applied to each data set 5 different times. The reported results are the average over those 50 runs. Two different experiments are conducted with each data set:

- Using all of the training data set (two-class classification problem)
- Using only the normal training samples (anomaly classification).

RAIN is executed using a decay term of $\Delta(G) = 0.001$, a merging distance of $\epsilon = 1e - 4$, $f(x) = e^{-x^2}$ as interaction function, and a maximum number of iterations of $\sqrt{n}$, ($M = 1$), where $n$ is the number of data points in the training data set.

#### 6.5.5.2 Analysis of Results

Table 6.4 shows the performance reached by RAIN in the tested data sets for both variations: using all training samples (Full-RAIN) and only using normal samples (Anomaly-RAIN). A value of $a \pm b$ indicates an average value of $a$ with standard deviation of $b$.

According to these results Anomaly-RAIN performs better than Full-RAIN. This behavior is expected since Anomaly-RAIN is creating clusters only for the normal class, while Full-RAIN is creating clusters for
Table 6.4: Performance reached by RAIN in two-class machine learning data sets. (a) Using all of the data samples, (b) Using only normal samples.

<table>
<thead>
<tr>
<th></th>
<th>Full-RAIN</th>
<th>Anomaly-RAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy %</td>
<td>G</td>
</tr>
<tr>
<td>BREAST</td>
<td>84.98±11.78</td>
<td>0.498±0.002</td>
</tr>
<tr>
<td>PIMA</td>
<td>65.78±1.64</td>
<td>0.204±0.003</td>
</tr>
<tr>
<td>HEART</td>
<td>58.07±6.35</td>
<td>0.488±0.002</td>
</tr>
</tbody>
</table>

Both classes. If samples of both classes (normal and abnormal) are used, RAIN can cluster samples from both classes into the same cluster because RAIN is creating the clusters without using the class label of the data samples. However, Full-RAIN was able to generate a good collection of clusters for each class and its performance was similar to the one using only normal samples (Anomaly-RAIN).

Notice that for each data set, the initial interaction strength (G) calculated by RAIN was similar in all 50 runs. In any case, the standard deviation was lower than 0.02. These results indicate that the setting of the initial interaction strength is a robust process and suggest that a behavior emerges from the random data point interaction process. Such emergent behavior allows the conformation of almost the same clusters. As shown, the standard deviation in the number of clusters is lower than 2 for all three data sets using Full-RAIN and Anomaly-RAIN. Moreover, the average number of clusters is lower than 7 for Full-RAIN and lower than 5 for Anomaly-RAIN. Thus, RAIN generates a compact representation of the normal (and abnormal) classes.

### 6.5.5.3 Comparing RAIN and T-FD

Table 6.5 compares the performance reached by RAIN and the Tuning of Fuzzy rule detectors (T-FD) mechanism proposed in chapter 5 in the anomaly classification problem. As shown, the performance of RAIN is similar to the one reached by T-FD. Moreover, the number of prototypes (clusters) generated by RAIN is much lower than the number of prototypes (Rule detectors) generated by T-FD. While RAIN generates between 3 and 5 clusters in average, T-FD generates between 30 and 45 rule detectors in average. In this way, RAIN scores in prediction time and interpretability.
Table 6.5: Performance comparison of RAIN and T-FD in the anomaly classification problem.

<table>
<thead>
<tr>
<th></th>
<th>RAIN</th>
<th>T-FD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy %</td>
<td>No. Clusters</td>
</tr>
<tr>
<td>BREAST</td>
<td>87.30±5.73</td>
<td>4.88±1.55</td>
</tr>
<tr>
<td>PIMA</td>
<td>65.36±1.45</td>
<td>4.60±1.23</td>
</tr>
<tr>
<td>HEART</td>
<td>62.82±5.78</td>
<td>3.74±0.90</td>
</tr>
</tbody>
</table>

6.6 Experimentation with Intrusion Detection Data Sets

6.6.1 Darpa 99

6.6.1.1 Experimental Settings

The score of the system of clusters generated using the training data set is calculated as the classification accuracy over the testing set. This process is repeated 10 times and the reported results are the average of these 10 runs. Each RAIN algorithm is executed using a decay term of $\triangle(G) = 0.001$, a minimum clusters size of $\alpha = 0.001$, a merging distance of $\varepsilon = 1e - 4$, $f(x) = e^{-x^2}$ as interaction function, and a maximum number of iterations of $2\sqrt{n}$, ($M = 2$), where $n$ is the number of data points in the training data set.

6.6.1.2 Results and Analysis

Figure 6.19 compares the ROC curve generated by RAIN and T-FD. As can be seen, the difference between the performance reached by RAIN and T-FD is minimal. However, the ROC curve generated by RAIN dominates the ROC curve generated by T-FD.

Table 6.6 presents the average detection rate and false alarm rate reached by RAIN and T-FD along with the average number of prototypes (clusters and fuzzy rule detectors) generated when the false alarm rate is set to 2.0%.

Table 6.6: Comparative detection rate of T-FD and S-FD when false alarm is set to 3%.

<table>
<thead>
<tr>
<th></th>
<th>FA %</th>
<th>DR %</th>
<th>Prototypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAIN</td>
<td>2.00</td>
<td>94.69±0.00</td>
<td>1.00±0.00</td>
</tr>
<tr>
<td>T-FD</td>
<td>2.00</td>
<td>93.94±0.87</td>
<td>11.5±1.18</td>
</tr>
</tbody>
</table>

As expected, the performance of RAIN and T-FD is very similar. However, these results indicate that RAIN generates a simpler model than T-FD. While T-FD generates around 11 fuzzy rule detectors, RAIN generates a single cluster.
6.6.2 KDDCup99

6.6.2.1 Experimental settings

A collection of 2000 normal data records are drawn from the data set and used as training data set while the remaining part of the data set is used as a testing data. The score of the trained classifier is calculated as the classification accuracy over the testing set. This process is repeated 10 times and the reported results are the average of these 10 runs. Each RAIN algorithm is executed using a decay term of $\triangle(G) = 0.001$, a minimum clusters size of $\alpha = 0.001$, a merging distance of $\varepsilon = 1e - 4$, $f(x) = e^{-x^2}$ as interaction function, and a maximum number of iterations of $2\sqrt{n}$, ($M = 2$), where $n$ is the number of data points in the training data set.

6.6.2.2 Results and Analysis

Figure 6.20 shows the ROC curve generated by RAIN on the KDDCup99 data set and compares it against the ROC curve generated by T-FD and EFR (Evolution of Fuzzy Rule technique proposed in chapter 4). Notice that the ROC curve generated by RAIN practically dominates the ROC curves generated by T-FD and EFR. Specifically, RAIN is able to detect more attacks than T-FD and EFR when the false alarm rate is set to a value lower than 5%.

Figure 6.19: ROC curves generated by RAIN and T-FD on the Darpa99 data set.
Figure 6.20: ROC curves generated by RAIN, T-FD and EFR on the KDDCup 99 data set.

Table 6.7 compares the detection rate reached by RAIN, T-FD and the (EFR) when the false alarm rate is fixed to 1% and 8.4% along with the number of prototypes generated by each technique. Clearly, RAIN outperforms both T-FD and EFR when the false alarm is set to 1%.

If each attack is considered independently, RAIN generates the ROC curves shown in Figure ??.

As expected, the behavior of the ROC curve considering only DoS attack is almost the same as considering all the attacks. It is due to the fact that DoS includes more than 99% of the attacks.

Table 6.8 shows the classification accuracy of RAIN, T-FD, and EFR when the false alarm rate is set to 8.4%. A false alarm rate of 8.4 is selected in order to compare the detection rate of the proposed methods.

Notice that, the detection rate of attacks different to DoS reached by RAIN is not so bad taking into account the proportion of samples belonging to those attacks. For example, RAIN is able to detect 63.61% of
Detection Rate (%)  
False Alarm Rate (%)  

Figure 6.21: ROC Curve generated by RAIN for each different attack

Table 6.8: Comparative performance of RAIN, T-FD and EFR.

<table>
<thead>
<tr>
<th>Class</th>
<th>RAIN</th>
<th>T-FD</th>
<th>EFR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>91.60%</td>
<td>91.60%</td>
<td>91.60%</td>
</tr>
<tr>
<td>U2R</td>
<td>61.08%</td>
<td>51.24%</td>
<td>55.77%</td>
</tr>
<tr>
<td>R2L</td>
<td>19.06%</td>
<td>40.30%</td>
<td>66.45%</td>
</tr>
<tr>
<td>DOS</td>
<td>98.88%</td>
<td>99.55%</td>
<td>96.52%</td>
</tr>
<tr>
<td>PRB</td>
<td>63.61%</td>
<td>74.93%</td>
<td>89.45%</td>
</tr>
</tbody>
</table>

the Prb attacks. Considering the KDDCup problem as multi-class problem, EFR performs better than RAIN and T-FD while T-FD performs a little better than RAIN.

6.7 Summary

This chapter presented a new robust clustering technique originated on the concept of Gravitational Law. For a $k$-dimensional data set with $n$ data points, each data point is considered as an object in the $k$-dimensional space with mass equal to 1. Each point in the data set is moved according to the gravitational field of another data point randomly selected. Then, a cooling mechanism is used in order to eliminate the big-crunch effect – all points assigned to a single cluster.
Such clustering algorithm was extended to any decreasing function of the distance between points. An estimation of the maximum distance between closest points was developed in order to reduce the effect of the size of the data set on the clustering technique. Also, an heuristic process based on a binary search was developed for setting the initial interaction strength (gravitational constant).

Experiments with well known synthetic data sets and intrusion detection data set were performed in order to show the applicability of the proposed approach. The results indicate that RAIN performs well in a variety of problems including anomaly detection and intrusion detection.

The next chapter introduces a technique based on coupled map lattices that allow one to integrate all the techniques developed in this and previous chapters. The purpose of that new technique is to get the best properties of each technique in a single one by distributing the (anomaly) classification among several simple entities.
Chapter 7

CORAL: Cell Organized Recognition Algorithms

7.1 Introduction

This work has introduced three different techniques for data mining and applied them to some well known machine learning and intrusion detection data sets. Each of these techniques has advantages and disadvantages. This chapter presents a technique based on coupled map lattices for integrating the previously developed techniques into a single (anomaly) classification system [73, 67, 101]. The system is designed in such a way that it is robust, distributed, and has a higher performance than each one of the integrated techniques. In particular this chapter:

- Develops a fuzzy class binarization technique based on the concept of coupled map lattices,
- Introduces a boosting technique using coupled map lattices, and
- Integrates the proposed techniques.

7.2 Fuzzy Class Binarization using Cell Organized Recognition Algorithms (CORAL-FCB)

CORAL-FCB combines both fuzzy unordered and fuzzy round robin class binarization schemes into one binarization scheme. In this way, CORAL-FCB transforms a $M$-class problem into $M^2$ two-class problems
by defining two classifiers $\text{classifier}_{i,j}$ for each pair of different classes $(i,j)$ and one classifier $\text{classifier}_i$ for each different class. Every classifier $\text{classifier}_{i,j}$ is trained with only samples of class $i$ and class $j$ by considering class $i$ as the negative class (class $j$ is considered the positive class). Therefore, CORAL-FCB trains a classifier $\text{classifier}_{i,j}$ following the round robin training strategy. Now, every classifier $\text{classifier}_i$ is trained with samples of class $i$ as negative and samples of other classes as positive. Then, a classifier $\text{classifier}_i$ is trained following the unordered training strategy.

Notice that it is possible to organize the two-class classifiers in a two dimensional lattice $x$, such that, $x(i,j) = \text{classifier}_{i,j}i \neq j$ and $x(i,i) = \text{classifier}_i$. Such a lattice has the following properties:

- A classifier in row $i$ considers class $i$ as negative.
- A classifier in column $j$ (exception done with the diagonal element) considers class $j$ as positive.
- A classifier in the diagonal position $i$ considers class $i$ as negative.

Therefore, elements in the same row or in the same column are correlated. Using these properties, it is possible to define the neighborhood of a site $x(i,j)$ (two-class classifier) as follows (equation 7.1):

$$N_{x(i,j)} = \{x(i,k)|k = 1, 2, ..., M \text{ and } k \neq j\} \cup \{x(k,j)|k = 1, 2, ..., M \text{ and } k \neq i\} \quad (7.1)$$

Sites in the same row or column of a given site belong to the neighborhood of such a site. The size of the neighborhood is $2(M-1)$ where $M$ is the number of classes. Figure 7.7 shows the neighborhood definition for a three-class classification problem.

In order to use this lattice for classifying a data sample $\text{sample}$, some state concept and dynamics should be associated with it. The state of a site $x(i,j)$ at any time $t \geq 0$ indicates the confidence degree that such a site has on classifying the data sample into class $i$. The initial state of site $x(i,j)$ is defined as the membership degree of the sample to class $i$ generated by the classifier associated to such a site, see Equation 7.2:

$$x_0(i,j) = \begin{cases} 
\mu_{i,\text{classifier}_i}(\text{sample}) & i = j \\
\mu_{i,\text{classifier}_{i,j}}(\text{sample}) & i \neq j 
\end{cases} \quad (7.2)$$

Now, the state of a site $x(i,j)$ evolves on time according to equation 7.6:
Figure 7.1: CORAL-FCB neighborhood definition for a three-class classification problem.

\[ x_{t+1}(i, j) = (1 - \varepsilon) x_t(i, j) + \frac{\varepsilon}{2(M - 1)} \left[ \sum_{k = 1, k \neq j}^{M} f(x_t(i, k)) + \sum_{k = 1, k \neq i}^{M} f(1 - x_t(k, j)) \right] \quad (7.3) \]

where \( f(x) \) is the logistic function, see equation 2.2. We divided the neighborhood into two groups, row-neighborhood and column-neighborhood, because their state is considered different by the site in evolution: On one hand, the state of a site of the row-neighborhood is used directly because such site considers class \( i \) as negative, like site \( x(i, j) \); On the other hand, the state of a site in the column neighborhood is negated (fuzzy negation) because such site considers class \( j \) as negative, opposite to the site \( x(i, j) \). Finally, the classification of a data sample is determined by a defuzzy operator that is applied to the state of the sites in the diagonal after some number of iterations \( MAX \) of the CML. Algorithm 17 shows the CORAL-FCB classification.
Algorithm 17 CORAL-FCB Classification Algorithm

CLASSIFY( classifier[1..M][1..M], MAX, sample )
1 for i = 1 to M do
2 for j = 1 to M do
3 \( x[i][j] = \text{Fuzzy}(\text{classifier}[i][j], i, \text{sample}) \)
4 for i = 1 to MAX do
5 \( x = \text{Iterate}_\text{CML}(x) \)
6 for i = 1 to M do
7 vote[i] = \( x[i][i] \)
8 return \( \text{Defuzzy}(\text{Normalize}(\text{vote})) \)

7.2.1 Experimentation

In order to test CORAL-FCB, the evolution of fuzzy rules with fuzzy set tuning mechanism proposed in chapter 4 (EFR) is used for evolving a fuzzy rule for each two-class classification problem generated with CORAL-FCB.

7.2.1.1 Experimental Settings

Experiments are performed in the multi-class machine learning data sets introduced in chapter 2. The CORAL-FCB, fuzzy unordered class binarization (FUCB), and fuzzy round robin class binarization (FRRCB) are tested using Heaps encoding scheme, tuning of fuzzy sets, and Average-Restricted Sum as fuzzy logic operators. The two-class evolutionary algorithm (which evolves the fuzzy rule associated with each two-class problem) is executed for 100 iterations using 100 individuals as population and VLSPX, SBM, ADD and DEL as genetic operators. A 10 folding cross-validation technique is applied to each data set 5 different times. The reported results are the average over those 50 runs. A coupling parameter of \( \varepsilon = 0.5 \) is used in order to give more importance to the site than to its neighbors, and the CORAL-FCB classifier is iterated 100 periods of time using the logistic function with parameter \( \alpha = 1.5 \).

7.2.1.2 CORAL-FCB Dynamics

One of the 50 classifiers generated by CORAL-FCB for the WINE data set was selected in order to plot the classifier site’s state through the time when a data sample was presented to such classifier. A sample per class was selected from the testing data set. Figure 7.2 shows the evolution of the state for each site of the CORAL-FCB classifier.

Since the logistic function is not introducing chaos to the CML, the state of a site has a non-chaotic behavior with a two-state limit orbit. This behavior indicates that CORAL-FCB is able to reach a stable condition. As expected, one of these states is approximated when CORAL-FCB executes an even number of
Figure 7.2: CORAL-FCB Dynamics for a data sample belonging to class 1 in the WINE data set.
iterations, while the other state is approximated when CORAL-FCB executes an odd number of iterations. Similar behavior is observed for the two data samples belonging to class 2 and class 3. Figure 7.3 shows the separated evolution of the limit orbit states for each diagonal site and each of the three data samples used.

![Figure 7.3: Separated dynamics of two-states that define the limit orbit for the diagonal sites. First row shows the even state (state reached by CORAL-FCB in an even number of iterations), while second row shows the odd state. Column 1 (2 and 3) when CORAL-FCB is classifying a data sample of class 1 (2 and 3 respectively).](image)

Notice that a better classification is obtained when the number of iterations executed by the CORAL-FCB classifier is an even number. Consider the data sample of class 1. In this case, site $x(1, 1)$ has higher value than sites $x(2, 2)$ and $x(3, 3)$ when the number of iterations is even, but has lower value when it is odd. This behavior is due to the coupling function used (logistic function) and the restriction of the values generated by each two-class fuzzy classifier (membership values belong to the interval $[0, 1.0]$). Therefore, a CORAL-FCB classifier should be allowed to execute an even number of iterations in order to obtain a good classification rate. Figure 7.4 shows the evolution of the average accuracy reached by CORAL-FCB on the 10 different runs for the tested data sets, when using an even number of iterations.

As shown, CORAL-FCB does not requires many iterations in order to reach a good classification performance: around 20 iterations is good enough.
Figure 7.4: Evolution of the average performance reached by CORAL-FCB on the tested data sets. (a) IRIS, (b) WINE and (c) GLASS.

### 7.2.1.3 Comparing CORAL-FCB against Fuzzy Round Robin and Fuzzy Unordered Binarization Schemes

Table 7.1 shows the performance (accuracy) reached by using CORAL-FCB after 20 iterations, Fuzzy Unordered binarization (FUB) and Fuzzy Round Robin Binarization (FRRB). In general, CORAL-FCB performs better than FUB and FRRB for these data sets.

<table>
<thead>
<tr>
<th></th>
<th>CORAL-FCB</th>
<th>FUB</th>
<th>FRRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIS</td>
<td>95.33±5.37</td>
<td>96.13±4.24</td>
<td>95.06±5.30</td>
</tr>
<tr>
<td>WINE</td>
<td>96.04±5.09</td>
<td>94.19±5.22</td>
<td>94.39±5.83</td>
</tr>
<tr>
<td>GLASS</td>
<td>54.60±9.44</td>
<td>53.66±9.26</td>
<td>53.82±9.07</td>
</tr>
</tbody>
</table>

### 7.2.1.4 Robustness

In order to determine the robustness of the proposed approach, different levels of damage are introduced into the classifier generated by CORAL-FCB for the WINE data set. We simulate the failure of a site by initializing it with the value 0.5 instead of using the fuzzy classifier associated to the site. In this way, a site \(x(i, j)\) is considered failing when its associated classifier is not able to determine the class of any data sample.

We simulated the CORAL-FCB classifier when a single site was failing: (1,1), (2,2), (3,3), and (2,1). Figure 7.5 shows the evolution of the average performance reached by the failing CORAL classifiers.

As expected the performance of the classifier is not strongly affected when one site is failing. In some cases, the performance of the classifier is the same or better than the original classifier. Moreover, the worst performance reached by these classifiers after 20 iterations was 94.72 (failing site : (2,1)).
Figure 7.5: Evolution of the accuracy of CORAL-FCB with one or none site failing.

Also, we determine the performance of the CORAL-FCB classifier when different groups of three sites are failing: diagonal sites, same row sites, same column sites and sites (1,2), (2,3) and (3,1). Figure 7.6 shows the evolution of the average performance reached by these failing CORAL-FCB classifiers.

Figure 7.6: Evolution of the accuracy of CORAL-FCB when 3 sites are failing. (a) Failing diagonal and rows, (b) Failing columns and sites {(1,2), (2,3), (3,1)}

Again, CORAL-FCB is able to perform well when the amount of damage introduced is 33.3%. The worst performance reached by CORAL-FCB after 20 iterations was 85.88% of accuracy when the first row of sites is failing. Otherwise, the accuracy reached was higher than 90%.
7.3 **Coral-B: Boosting a Two-class Classification Technique**

*Coral-*B distributes the classification task among several interconnected two-class classifiers, each classifier trained with (possible) different data sets. This task distribution is done in order to:

- Reduce the training time – each classifier can be generated in a different computer,
- Increase the robustness of the classification system, and
- Increase the accuracy of the system.

*Coral-*B organizes the classifiers in a two dimensional CML of size $N \times M$ where the neighborhood of site $x(i,j)$ is defined as:

$$N_{x(i,j)} = \{x(i,j+1), x(i,j-1), x(i-1,j), x(i-1,j)\}$$ (7.4)

Figure 7.7 shows the *Coral-*B neighborhood definition for a CML of size $3 \times 3$.

![Figure 7.7: Coral-B neighborhood definition for a CML of size 3 x 3.](image)

The state of a site $x(i,j)$ on any time $t \geq 0$ indicates the confidence degree that such site has on classifying a data sample *sample* into the negative class. The initial state of site $x(i,j)$ is defined as the membership degree of the sample to the negative class generated by the classifier associated to such site, see Equation 7.5:

---

1Sites that are on the border are connected with sites that are in the opposite border in the same row (column).
Algorithm 18 CORAL-B Classification Algorithm

\textbf{CLASSIFY(} \textit{classifier}[1..M][1..M], MAX, \textit{sample} \textbf{)}

\begin{align*}
1 & \text{ for } i = 1 \text{ to } M \text{ do} \\
2 & \text{ for } j = 1 \text{ to } M \text{ do} \\
3 & \quad x[i][j] = \text{Fuzzy}(\text{classifier}[i][j], i, \text{sample}) \\
4 & \text{ for } i = 1 \text{ to } MAX \text{ do} \\
5 & \quad x = \text{ITERATE}_\text{CML}(x) \\
6 & \text{ for } i = 1 \text{ to } M \text{ do} \\
7 & \quad \text{vote}[i] = x[i][i] \\
8 & \text{return Defuzzy( Normalize(\text{vote} ) )}
\end{align*}

\begin{equation}
x_0(i, j) = \mu_{-,\text{classifier}_{i,j}}(\text{sample}) \tag{7.5}
\end{equation}

Now, the state of a site \( x(i, j) \) evolves on time according to equation 7.6:

\begin{equation}
x_{t+1}(i, j) = (1 - \varepsilon) x_t(i, j) + \\
\tilde{\xi} \left[ f(x_t(i, j - 1)) + f(x_t(i, j + 1)) + f(x_t(i - 1, j)) + f(x_t(i + 1, j)) \right] \tag{7.6}
\end{equation}

where \( f(x) \) is the logistic function, see equation 2.2. Finally, the classification of a data sample is determined by a defuzzy operator that is applied to the state of the sites in the diagonal after some number of iterations \( MAX \) of the CML. Algorithm 18 shows the CORAL-B classification.

### 7.3.1 Experimentation

Four different experiments were carried on to determine the effectiveness of CORAL-B:

1. Boosting the evolution of fuzzy rules with fuzzy set tuning mechanism proposed in chapter 4 (EFR). Each site will have associated an EFR classifier that will discriminate between the positive and negative class.

2. Boosting the fuzzy self/non-self discrimination technique with fuzzy set tuning mechanism proposed in chapter 5 (T-FD). Each site will have associated a set of fuzzy rule detectors that will discriminate the positive class.

3. Boosting the clustering algorithm proposed in chapter 6 (RAIN). Each site will have associated a set of cluster prototypes that will discriminate the negative class.
4. Boosting a combination of the above three approaches (EFR, T-FD, and RAIN). Some sites will have associated an EFR classifier, while others a set of fuzzy rule detectors (T-FD) and the remaining a set of cluster prototypes (RAIN). This experiment is performed in order to show the integration capability provided by CORAL-B.

7.3.1.1 Experimental Settings

Experiments are performed on the two-class machine learning data sets introduced in chapter 2. Each technique used the parameters introduced in their respective chapter. A $3 \times 3$ CORAL-B lattice is used in all the experiments. An special 10 folding cross-validation technique is applied to each data set. In this way, each of the 9 sites (the classifier associated with such site) is trained with 1 of the 10 groups, and the remaining group is used as testing set. It is done 5 different times. The reported results are the average over those 50 runs.

A coupling parameter of $\varepsilon = 0.5$ was used in order to give more importance to the site than to its neighbors and the CORAL-B classifier was iterated by 20 periods of time using the logistic function with parameter $\alpha = 1.5$.

Figure 7.8 shows the distribution of techniques used by CORAL-B-4 (combination of all techniques).

![Figure 7.8: Distribution of Techniques used by CORAL-B-4 (combination of all techniques).](image)

7.3.1.2 Analysis of Results

Table 7.2 shows the performance (accuracy) reached by CORAL-B after 20 iterations using EFR, T-FD, RAIN, and all of them.

Table 7.2: Performance of CORAL-FCB, Fuzzy Unordered (FUB) and Fuzzy Round Robin (FRRB).

<table>
<thead>
<tr>
<th></th>
<th>BREAST</th>
<th>HEART</th>
<th>PIMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORAL-B-1 (EFR)</td>
<td>97.68±1.87</td>
<td>92.81±5.10</td>
<td>77.79±4.57</td>
</tr>
<tr>
<td>CORAL-B-2 (T-FD)</td>
<td>97.80±2.12</td>
<td>69.85±9.34</td>
<td>67.42±1.92</td>
</tr>
<tr>
<td>CORAL-B-3 (RAIN)</td>
<td>83.94±9.82</td>
<td>61.56±9.79</td>
<td>65.81±3.04</td>
</tr>
<tr>
<td>CORAL-B-4 (All)</td>
<td>95.74±2.36</td>
<td>68.30±6.67</td>
<td>67.51±2.78</td>
</tr>
<tr>
<td>EFR</td>
<td>94.85±2.41</td>
<td>78.96±8.26</td>
<td>74.21±5.63</td>
</tr>
<tr>
<td>T-FD</td>
<td>92.85±3.61</td>
<td>62.44±9.04</td>
<td>65.47±4.66</td>
</tr>
<tr>
<td>RAIN</td>
<td>87.30±5.73</td>
<td>62.82±5.78</td>
<td>65.36±1.45</td>
</tr>
</tbody>
</table>
When CORAL-B is used with EFR, it improves the accuracy of the classifier system in at least 2.8% for all three data sets. Moreover, for the Heart data set, the accuracy is increased in almost 14%. Similar behavior is observed when CORAL-B is used with T-FD. In this case the accuracy increased in at least 1.9% with an increase around 7% for the Heart data set. However, this behavior is not observed in the case of CORAL-B when using RAIN. In this case CORAL-B is better only in the Pima data set for a small margin (less than 0.5%). For the Breast data set CORAL-B reduces the accuracy by almost 3.5%. It is possible that RAIN is not able to generate good cluster prototypes from the small amount of information it is training on – each site is trained with 10% of the data set meaning around 45 normal (negative) samples.

7.4 Summary

A technique for class binarization based on Coupled Map Lattices and fuzzy logic that uses evolution of fuzzy rules as proof of concept was proposed. The results indicate that the performance of the proposed approach (CORAL-FCB) is better than other binarization schemes. Moreover, the performance of the classifier is not strongly affected when different levels of failures are introduced to the lattice.

Also, a boosting technique based on Coupled Map Lattices and fuzzy logic was introduced. The results indicate that it improves the accuracy and robustness of the classifier while reducing the training time. It was shown that several different training classification techniques can be used and integrated into a CORAL-B classification technique.
Chapter 8

Conclusions and Future Work

Solving the problem of intrusion detection is not an easy task since it is not well defined—there is not a precise definition of normal/abnormal behavior. Many techniques have tried to solve the problem of intrusion detection, but all of them have issues in terms of scalability, robustness, and/or sensitivity to parameters. That is also true for soft computing techniques regardless of being natural candidates for solving not well-defined problems like the intrusion detection. Many soft computing techniques require a large number of parameters that should be tuned in order to achieve a good performance. Some techniques do not scale well when applied to huge amounts of collected data representing normal/abnormal behavior.

Since the problem of intrusion detection can be seen as a classification problem, it is possible to integrate different data mining techniques to find a solution for it. The main goal of the work presented in this dissertation was to define a set of soft data mining techniques (data mining techniques based on soft computing concepts) and to define a mechanism for integrating them in order to be able to solve the problem of intrusion detection.

8.1 Main Contributions

Three different soft data mining techniques were introduced in this work, one for classification, other for anomaly classification, and the last for clustering, and a technique for integrating them was also proposed. Specifically:

- This work defined a general framework for evolving fuzzy rule-based classifiers and applied it to solve some classification and intrusion detection problems. This framework combines fuzzy logic, evolutionary algorithms and class binarization concepts. A classification problem is divided in several
two-class classification problems using a fuzzy version of a class binarization scheme. Then, an evolutionary algorithm is used to evolve a fuzzy rule based classifier. Finally, the evolved fuzzy rule based classifiers are combined following the class binarization scheme.

- Fuzzy versions of the unordered and round robin class binarization techniques were proposed. Surprisingly, the results indicated that there is not a binarization technique that can be considered better. Both techniques reached similar performance in the tested data sets.

- A generic evolutionary algorithm for evolving fuzzy rules was proposed. In particular,
  - An abstract fuzzy rule encoding scheme was developed. This encoding scheme covers several encoding mechanisms proposed in the literature and allows one to perform a fair comparison between them. According to the results obtained, the encoding proposed in this dissertation (Heaps encoding) performs better than all of the tested encoding schemes. This behavior was expected since the expressiveness of Heaps is higher than normal forms encoding and it is less disruptive than the encoding based on precedence of operators.
  - These encoding mechanisms were extended in such a way that tuning of fuzzy sets is allowed. In this way, not only the fuzzy rule is evolved, but the fuzzy space that is associated with each attribute is evolved as well. This reduces the sensitivity of the approach to the criteria of the user since the user does not have to predefine the fuzzy linguistic values that each attribute has associated.
  - Special genetic operators for the above encoding were presented. The set of operators includes variable length crossover, gene addition, and gene deletion. Also, a disruption concept for fuzzy rule encodings was proposed in order to study the effect of genetic operators on different fuzzy rule encoding schemes. As mentioned above, the genetic operators introduced less disruption on the Heaps encoding than in the encoding based on the precedence of operators. These operators introduced less disruption in the normal forms encoding than in the Heaps encoding.
  - The concept of a fuzzy confusion matrix was introduced in order to define the fitness function. This fuzzy confusion matrix is a generalization of the concept of confusion matrix and allows the evolutionary algorithm to smoothly determine the fitness of an individual.
  - In order to reduce the amount of parameters in the evolutionary algorithm, this work proposed a hybrid adaptive evolutionary algorithm (HAEA) that adapts the genetic operator rates when searching for the solution. In this way, different genetic operators can be used.
and no rates need to be associated with them. This has a clear advantage when evolving fuzzy rules since four different genetic operators are used for the evolutionary algorithm. The results obtained in well known optimization functions were promising and indicated that HAEA is able to find the optimal solution to problems that generational and steady state genetic algorithms were no able to find.

- This work compared the performance of the proposed approach when different fuzzy logic operators were used. It was shown that there is not a significant between different combinations of fuzzy logic operators.

- This work extended the artificial immune system for anomaly detection with fuzzy logic. Experiments with machine learning and intrusion detection data sets showed the applicability of the proposed approach. In particular,

  - The interpretability of the fuzzy rules generated by the original fuzzy self/non-self discrimination technique proposed in [71] was improved by using structured genetic algorithms and fuzzy set tuning. The results indicated that the number of fuzzy rule detectors was reduced to more than a half of the original technique while the number of required evolutions was drastically reduced (close to the 10%). Moreover, these results indicate the structured genetic algorithm approach and the fuzzy set tuning mechanism performed almost the same when the size of the training data set is small, but the fuzzy set tuning mechanism outperforms the structured genetic algorithm when the data size is large.

  - A connection between Genetic Fuzzy Rule Based Classifiers and Fuzzy Self/Non-self discrimination of Artificial Immune Systems was established. It was shown that the last is an extension of the first when there are not samples of the abnormal class in the training data set and a niching technique is introduced for evolving different fuzzy rules. In this way, the fitness function of the fuzzy self/non-self discrimination technique was rewritten in terms of the fitness function of the genetic fuzzy rule based classifiers and the conjunctive normal form was used for evolving the fuzzy rule detectors.

  - A niching strategy for HAEA was designed in order to search for more than one good global solution. Such niching mechanism is a version of the Deterministic Crowding introduced by Mahfoud in [117]. It was shown that HAEA with deterministic crowding is able to solve multi-modal optimization problems, thus it is appropriated for evolving many fuzzy rules in a single run.
- This work introduced a clustering algorithm that uses gravitational concepts. This algorithm reduces the time complexity from cubic to super linear of the original one introduced by Wright in [161]. Also, the proposed clustering technique deals with noise and automatically determines the number of clusters. Also, this work extended the clustering technique in such a way that not only the gravitational force can be used, but any decreasing function of the distance between points can be used. Moreover, this work incorporated a fuzzy analysis to the clusters generated for reducing the sharp distinction between them. These techniques were applied to different synthetic data sets, machine learning data sets, and to some intrusion detection data sets. The results are promising. In particular,

- This work proposed a randomized version of the gravitational clustering algorithm based on the Gravitational Law and the second Newton’s motion Law. Experimental results showed that this clustering technique runs in super-linear $O(n\sqrt{n})$ respect to the size of the data set. It is done by simulating in a randomized way the dynamics of the gravitational system. For moving a data point it does not require the information of any other point in the system, but only the information of another single point randomly selected. As expected, this “small” modification in the simulation process reduces the time complexity by one order. A kind of cooling mechanism similar to the one in simulated annealing was introduced in order to eliminate the formation of a single cluster (big-crunch) and noisy information (robust clustering). Experimental results indicated that such a cooling mechanism was useful and allowed the clustering technique to remove noise and determine the number of clusters in the data set. Also, the results indicated that the clustering technique can be used with a portion of the data set (around 20%) while keeping the performance high.

- This clustering technique was extended in such a way that it uses different dynamic functions for simulating the system. Basically, three elements were considered: reducing the effect of the data set in the dynamics of the system, defining an interaction function, and setting automatically the initial interaction strength (gravitational force).

* In order to reduce the effect of the data set in the dynamics of the system, a notion of maximum distance between closest point was introduced. This notion will allow the clustering algorithm to “normalize” the dynamics of the system according to the data set. An approximation to the maximum distance between closest points was proposed and the results indicate that it is a good approximation.
Using the “normalized” data set it was possible to introduce different interactions function between data points. In this paper two different functions were used. One function using concepts of density functions and the other considering the original approach.

A kind of binary search process was implemented in order to determine the initial interaction strength (gravitational force). A notion of “good” clustering performance was introduced in order to determine the correctness of the initial interaction strength. The results indicated that such a search mechanism was able to find appropriated values for all the tested data sets. Surprisingly, for a given data set, the value found by this approach remained relatively constant through.

This work integrated the above three techniques in one system. Each technique was associated with the basic computational unit of a cell organized intrusion detection tool (CORAL). Such a tool is able to solve the (anomaly) classification problem in a distributed fashion. Moreover, this technique is robust to unit (cell) failures. Although, it was not tested with intrusion detection data sets, the performed experiments showed the applicability of the approach to solve the intrusion detection problem. In particular,

– A fuzzy class binarization based on CORAL was developed. This fuzzy class binarization performed better than both fuzzy unordered and round robin class binarizations. Moreover, the results indicated that it is robust to cell damage.

– CORAL was used as a boosting technique for the three previously proposed approaches. The results indicated that this technique performed better than each simple technique. It also reduced the training time since each cell can be trained with a portion of the data set in a different computer. The three techniques were integrated using the CORAL architecture. Although the results reported were biased for the classification technique, it showed that it is possible to create a robust technique that utilizes different techniques. In this way, such techniques should provide a fuzzy normalcy level instead of a crisp one.

8.2 Future Work

Although all the proposed techniques have shown their applicability to solve some classification, anomaly classification, clustering, and intrusion detection problems, there are some issues that can be addressed in future works:
• Extend the HAEA approach in order to eliminate the setting of population size and number of iterations. First, some criteria for adding and removing individuals from the population should be defined. It is possible to define some meta-operators that take into account the population diversity and adjust the population size accordingly. Second, the stop criteria can be adjusted in such a way that the evolutionary process reaches some level of diversity and/or convergence.

• Extend the Genetic Fuzzy Rule Based Classifier approach in order to include Pittsburgh approaches for evolving multiple fuzzy rules in a single run. It is possible to extend the crowding scheme introduced for self/non-self discrimination to any kind of positive expressions. The distance between fuzzy rules can be defined in a recursive manner taking into account possible repetitions of attributes and fuzzy sets. Another option is to extract not only the best rule, but a percentage of the best rules. Then such rules can be weighted according to the performance of the best fuzzy rule. A more interesting idea is to evolve within a single run the full fuzzy class binarization scheme. In this way, in a single run each two-class fuzzy rule classifier can be evolved while sharing information with each other. A possible encoding scheme should include the indices of the two-class classifier that an individual is discriminating.

• Extend the Fuzzy self/non-self discrimination technique to use any positive expression. The approach proposed in this work uses conjunctive normal forms. It will be interesting to see how this approach can deal with other type of encoding schemes. The notion of distance between fuzzy rules has to be extended as proposed in the previous bullet.

• Extend the clustering algorithm in such a way that the decay term and the size extraction parameters can be removed. According to the experimentation performed, it is possible to define some heuristic for setting the decay term automatically. The process is similar to the one used for setting the initial interaction strength. For the size extraction parameter it is possible to use cluster validation techniques that have been reported in the literature, see [162, 164, 16, 135, 131].

• Analyze the effect of introducing chaos in the CORAL approach. It is important to determine if introducing chaos increases the performance of the proposed approach. This will give some clue to possible extensions of the CORAL approach to (semi) on-line learning. Basically, each cell will have one technique that will be trained according to the global response of CORAL to the presented data sample. Some feedback from the user can be introduced in the form of a cell too.
References


