

Djillali Liabes University

Department of Computer Science

Date : October, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Romaissaa Mazouni

ENTITLED

**A GENERIC GENETIC PROGRAMMING(GGP) TO
GENERATE AUTOMATICALLY DEFINED FUNCTIONS
(ADFs) : application in datamining and information retrieval**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

Thesis Advisor Prof. Abdel-
latif Rahmoun

Chairman of Department
Dr.Soufiane Boukli Hacene

Thesis	Reader
Prof.Mohammed	Kamel
Feraoun	

Thesis	Reader
Prof.Mohammed	Ben-
mohammed	

Thesis	Reader	Dr.Soufiane
Boukli	Hacene	

Thesis	Reader	Dr.Reda Ad-
joudj		

**A GENERIC GENETIC PROGRAMMING(GGP)
TO GENERATE AUTOMATICALLY DEFINED
FUNCTIONS (ADFs) : application in datamining
and information retrieval**

By

Romaissaa Mazouni

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Departement of Computer Science at
Djillali Liabes University, 2016

Sidi Bel Abbès, Algeria

” It is possible to store the mind with a million facts and still be entirely uneducated ”
Alec Bourne

Acknowledgments

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this thesis. A special thanks to my supervisor , Pr. Abdellatif Rahmoun, whose help, stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

A special thanks goes to Dr,Nadia Menazela , who helped me to write the thesis and provided me with English support.

I would also like to acknowledge with much appreciation the help of Prof.Eric Hervet for his help, suggestions and corrections he gave me through all the years of my thesis. My completion of this project could not have been accomplished without the support of my beloved father Dr. Abdlghani Mazouni .

Finally, to my caring, loving, and supportive father, mother and husband , Father : my deepest gratitude. Your encouragement when the times got rough are much appreciated and duly noted. It was a great comfort and relief to know that you are there for me, Mother : the countless times you kept my son during our hectic schedules will not be forgotten. My heartfelt thanks.

I would like to thank the staff of LMPM laboratory, who gave the permission to use required machinery and the necessary material to complete my experiments.

Last but not least, i would like to express my sincere thanks to the jury members for accepting to judge this work :

prof. M.Benmohammed prof. K.Feraoun, Dr. S.Boukli Hacene, Dr. R.Ajdoudj.

To My Father Dr. Abdeghani Mazouni...

To My Mother Dr. Oumelkheir Ikhlef.....

To My Husband Prof. Smail Benbarek...

To My Beloved Son Nassim Benbarek....

Glossary of Important Terms

GP	Genetic Programming
GA	Genetic Algorithm
GGP	Grammar based Genetic Programming
GADS	Genetic Algorithm for Developing Software
GE	Grammatical Evolution
LOGENPRO	Logic Grammar Based Genetic Programming
AGGE	Automatic Generation of rule based classifiers
FuAGGE	Automatic Generation of Fuzzy rule based classifiers
CFG-GP	Context Free Grammar based Genetic Programming
CN2	Clark and Niblett's Algorithm 2
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
IREP	Incremental Reduced Error Pruning
REP	Reduced Error Pruning
BEXA	Basic EXclusion Algorithm
C4.5	Ross Quinlan Rule Inducer
PRISM	An Algorithm for Inducing Modular Rules
1RULE	One Rule Algorithm
FOIL	First Order Inductive Learner
m-FOIL	m-estimate First Order Inductive Learner
N	Set of Negative Instances
P	Set of Positive Instances
AQR	<i>A^qFamily of Algorithms</i>
mgc	Most General Conjunction
ARFF	Attribute Relation File Format
FARFF	Fuzzy Attribute Relation File Format
ID3	Iterative Dichotomiser 3
FuzzConRi	Fuzzy Conjunctive Rule Inducer
Fuzzy Bexa	Fuzzy Basic EXclusion Algorithm
FILSMR	Fuzzy Inductive Learning Strategy for Modular Rules
Fuzzy AL	Fuzzy Attributional Logic
Fuzzy CAL	Fuzzy Conjunctive Attributional Logic
Fuzzy PRISM	Fuzzy Algorithm for Inducing Modular Rules
Fuzzy CN2	Fuzzy Version of Clark and Niblett's algorithm 2

Grammatical Evolution to Automatically Evolve Computer Programs : Automating the Process of Designing Decision Rules Induction Algorithms

Romaissaa Mazouni

**Department of Computer Science
Djillali Liabes University
Sidi Bel Abbes, Algeria
2015**

ABSTRACT

One of the main and fundamental tasks of data mining is the automatic induction of classification rules from a set of examples and observations .

A variety of methods performing this task have been proposed and many comparative studies have been carried out in this field ; however the main common feature between these methods is that they are designed by humans, there have been a successful attempt to automatically design such methods using GGP. In this work we describe a different system that can evolve complete java program codes representing rule induction algorithms using the grammatical evolution technique that governs a Backus Naur Form grammar definition mapping to a program, in this system we will use as inputs to the mapper along with the Backus Naur Form grammar, integer strings representing potential solutions resulting from the initialiser component and Weka building blocks to facilitate the induction process and shorten the induced programs .

Rule induction algorithms evolved using this system can be of different complexities.

Data in real world applications are in most cases linguistic information that is ambiguous and uncertain. Hence, such data should be handled by fuzzy set representation schemes to increase expressiveness and comprehensiveness. Moreover, mining these data requires ways to generate automatically useful information/knowledge through a set of fuzzy rules.

In this work we propose an extension of this system is also proposed and tested. The new system is called FuAGGE which stands for Fuzzy Automatic Generator Genetic Expression. FuAGGE was conceived in a way that it can evolve rule based classifiers that generate fuzzy rule sets. FuAGGE uses a grammar based evolutionary technique. The grammar is expressed in the Backus Naur Form (BNF) and represents a fuzzy set covering method. The grammar is mapped into programs that are themselves implementations of fuzzy rule induction algorithms. FuAGGE has also been tested on a

benchmark of well-known datasets, and experimental results prove the efficiency of the proposed method. It is shown through comparison that our method outperforms most recent and similar, manual techniques. The system is able to generate rule induction algorithms specialized to specific domains, for example medical or biological data. It produces efficient rule models and achieves more accurate classification.

ABSTRACT

Une des principales tâches fondamentales et du domaine de la fouille de données est l'induction l'automatique des règles de classification à partir d'un ensemble d'exemples et d'observations.

Une variété de méthodes ont été proposées pour effectuer cette tâche et beaucoup d'études comparatives ont été menées dans ce domaine; cependant le principal caractéristique commune entre ces méthodes, ce est qu'ils sont conçus par les humains, il ya eu une tentative réussie de concevoir automatiquement ces méthodes utilisant GGP. Dans ce travail, nous décrivons un système différent qui peut évoluer des codes de programme JAVA complets représentant des algorithmes d'induction de règles en utilisant la technique de l'évolution grammaticale qui est basée sur une grammaire du type Backus Naur définition, dans ce système, nous allons utiliser comme entrées au mappeur la grammaire Backus Naur Form avec des chaînes d'entiers représentant les solutions potentielles résultant de l'initialisateur et les blocs de construction Weka pour faciliter le processus d'induction et de raccourcir le programmes induit.

Les algorithmes d'induction de règles, évolué utilisant ce système peuvent être de différentes complexités.

Les données dans les applications du monde réel sont dans la plupart des cas des informations linguistiques qui sont ambigu et incertain. Par conséquent, ces données doivent être traitées par les systèmes de représentation des ensembles flous pour augmenter l'expressivité et l'exhaustivité. En outre, l'exploitation minière de ces données nécessite des moyens pour générer automatiquement des informations utiles / connaissances grâce à un ensemble de règles floues.

Dans ce travail, nous proposons une extension de ce système. Le nouveau système est appelé FuAGGE qui signifie Fuzzy Automatic Generator Genetic Expression. FuAGGE a été conçu de façon à pouvoir évoluer des classificateurs à base de règles qui génèrent des ensembles de règles floues. FuAGGE utilise une technique évolutive en fonction de grammaire. La grammaire est exprimée sous la forme Backus Naur (BNF) et représente la méthode de couverture des ensembles flous. La grammaire est mappée dans des programmes qui sont eux-mêmes des implémentations d'algorithmes d'induction de règles floues. FuAGGE a également été testé sur une référence des jeux de données bien connues, et les résultats expérimentaux ont prouvé l'efficacité de la méthode proposée. Il est montré par la comparaison que notre méthode surpasse les techniques manuelles les

plus récentes. Le système est capable de générer des algorithmes d'induction de règles spécialisés à des domaines spécifiques, par exemple des données médicales ou biologiques. Il produit des modèles de règles efficaces et réalise une classification plus précise.

Table des matières

Glossary of Important Terms	viii
1 Introduction	1
1.1 Motivations	3
1.2 Aims and Objectives	4
1.3 Contribution	4
1.4 Thesis Organization	5
2 Genetic programming	6
2.1 Introduction	6
2.2 Grammar Based Genetic Programming Methods	13
2.2.1 Grammars	14
2.2.2 Genetic Algorithm for Developing Software (GADS)	16
2.2.3 Logic Grammar Based Genetic Programming (LOGENPRO)	18
2.2.4 Context Free Grammar based Genetic Programming (CFG-GP)	18
2.2.5 Grammatical Evolution (GE)	20
2.3 summary	23
3 Data Mining	25
3.1 Introduction	25
3.2 Data Mining and Analysis	27
3.3 Classification in Data Mining	28
3.4 Sequential Covering	31
3.4.1 Evaluation Metrics	33
3.4.2 Rule Pruning	35
3.5 Summary	37

4	Proposed System : Automatic Generator of Rule Base Classifiers (AGGE) and Results	39
4.1	Introduction	39
4.2	Individual Representation	41
4.3	Population Initialization	41
4.4	Mutation and Crossover	43
4.5	Fitness Function and Individual Evaluation	44
4.6	The Grammar	46
4.7	Results and Discussion	56
4.8	AGGE Classifiers Versus Manually Designed Classifiers	66
4.9	State of The Art	69
4.10	Summary	72
5	Automatic Induction of Fuzzy Rule Based Classifiers FuAGGE . . .	73
5.1	Introduction	73
5.2	Fuzzy Set Theory	74
5.3	Fuzzy Rule Set Learners	75
5.3.1	Fuzzy-AQR	76
5.3.2	Fuzzy-CN2 (FuzzConRi)	78
5.3.3	Fuzzy PRISM (FILSMR)	78
5.4	The System Settings	83
5.4.1	Fuzzy Set Covering Grammar	84
5.5	Results and Discussion	90
5.5.1	Data Pre-processing	92
5.5.2	Experimentation and Discussion	96
5.6	FuAGGE Classifiers Versus Manually Designed Classifiers	103
5.7	Summary	107
6	Conclusion	109
	Appendices	125
A	A Note on the Fu/AGGE System's Execution Time	126

Table des figures

2.1	Genetic Programming Research Field	8
2.2	Genetic Programming Pseudo Code	10
2.3	Genetic Programming Taxonomy	13
2.4	Genetic Algorithm for Developing Software (GADS) Process	17
2.5	Logic Grammar Based Genetic Programming (LOGENPRO) Process	19
2.6	Context Free Grammar based Genetic Programming (CFG-GP) Process	21
2.7	Grammatical Evolution Mapping Process	22
2.8	Grammatical Evolution (GE) Process	23
3.1	Data Mining Field Origins	26
3.2	The Sequential Covering Pseudo Code	32
3.3	The Learn One Rule Pseudo Code	32
3.4	A Prepruning Algorithm Pseudo Code	36
3.5	A Postpruning Algorithm Pseudo Code	36
3.6	TDP Pruning Method Pseudo Code	37
4.1	AGGE System Scheme	40
4.2	Individual Representation	42
4.3	Context Free Grammar Example	43
4.4	Crossover Ripple effect	44
4.5	The Mapping Process	46
4.6	The accuracy Computation	47
4.7	Modules of The Proposed System	55
4.8	An Example of a AGGE Generated Classifier	56
4.9	Pseudo Code of the Individual Represented in Figure 5.1	57
4.10	Frequencies of Refinement Terminal Symbols During the 20 Generations	63

4.11	Frequencies of Rules Evaluation Terminal Symbols During the 20 Generations	63
4.12	Frequencies of Rules Initialization Terminal Symbols During the 20 Generations	64
4.13	Frequencies of the Algorithm stopping criterion Terminal Symbols During the 20 Generations	65
4.14	Pseudo Code of the PRISM Algorithm	67
4.15	Pseudo Code of the RIPPER Algorithm	68
4.16	Rule Induction Algorithm Generate Using AGGE system	70
4.17	Rule Induction Algorithm Generate Using AGGE system	71
5.1	An Example of Membersshp Functions	75
5.2	Fuzzy-AQR Pseudo Code	77
5.3	Fuzzy-CN2 Pseudo Code	79
5.4	FindBestAntecedent Procedure Pseudo Code	80
5.5	Fuzzy-PRISM Pseudo Code	82
5.6	The Fuzzy Set Covering Grammar	91
5.7	A crisp data set (arff)	92
5.8	A fuzzy data set (farff)	93
5.9	Membership function construction algorithm	94
5.10	Frequencies of Refinement Terminal Symbols During the 60 Generations	99
5.11	Frequencies of Rules Evaluation Terminal Symbols During the 60 Generations	100
5.12	Frequencies of Alpha-Covering Values During the 60 Generations	101
5.13	Frequencies of the Beta-Belonging Values During the 60 Generations	101
5.14	Fuzzy-PRISM Pseudo Code	104
5.15	Rule Induction Algorithm Generate Using FuAGGE system	106
5.16	Rule Induction Algorithm Generate Using FuAGGE system	107

Liste des tableaux

4.1	Data Sets Description	58
4.2	Accuracy rates (%) Using Both Meta-Sets	60
4.3	Accuracy Rates (%) of Automatically Evolved Classifiers Using Both Meta-Sets	66
5.1	Data Sets Description	91
5.2	Accuracy rates (%) Using Both Meta-Sets	98
5.3	Accuracy rates (%) Using Both Meta-Sets (AGGE-classifiers versus FuAGGE- classifiers)	103
A.1	AGGE/FuGGE runtime for different experiments configurations	127

CHAPITRE 1

Introduction

Data mining have become a fashionable research field in the recent years, it is the process of extracting comprehensible accurate knowledge from the data that is collected and warehoused. Data mining can be seen as data analysis, it is used to extract models describing important classes or predict future data trends, it helps us to provide better understanding of large scalable data. Classification is a famous task of data mining that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the category of each record in the data, for example classification can be used to classify a tumor as benign or malignant.

Classification task begins with data sets in which each record has a known class assignment, there exists two types of classifications the binary one where the classes number is two (ex : tumor category classification as malignant or benign) and the multiclass classification where the number of values of the class attribute is greater than two (ex : classifying the type of iris plants as iris setosa, iris versicolor and virginica).

Built classification models can be represented in two ways depending on the classification tool or method used to generate these models, it can either be represented as a black box if we use neural networks for example or in a human comprehensible way such as decision rules. If the classification model comprehension is not needed and it does not matter if we know or not how the system is classifying records we go for black box representation methods. However if we need to know how the classification is performed we go for the human comprehensible methods.

In our thesis we choose the second representation type which is the human comprehensible one, more precisely we will work on the decision rule based classification. Rule based classification models are models containing a set of decision rules of the type If conditions Then consequence. There exist several methods to build rule based models and the one we will focus on in our work is the sequential covering strategy. Plenty of manually designed rule based classifiers following the sequential covering strategy exist in the literature such as PRISM [20], RIPPER[60], CN2[29]. a new approach have been introduced recently by Gisele Pappa [19] to design sequential covering algorithms and the particularity of this method is that it is automatic. Gisele proposed a system that can automatically design sequential covering algorithms by programming the steps that a trained human follow while design these type of algorithms.

It was a successful attempt that produced algorithms that are highly competitive with the manually designed ones. The system used a grammar that describes the overall structure of rule induction algorithms then this grammar was used along with genetic programming to evolve rule based classifiers.

The desire to automatically create computer programs dates back to the pioneering work of Samuel in the sixties when the term machine learning first appeared and by the early nineties a new area dedicated to automatic generation of computer programs appeared, this one was called Genetic Programming [19], another example dealing with automatic generation of computer programs besides Gisele's work is the work of Pfahringer and al which consisted of methods to automatically select suitable classification algorithms for a data set being mined, out of several algorithms [62]. The idea behind genetic programming is to give computers the ability to program them selves or each others and to give them the ability to automatically evolve solutions representing computer programs (code sequences). Building code sequences for complex real world problems from scratch can be a really tedious task therefore the use of new genetic programming method ap-

peared which is called Grammar based Genetic Programming, this method have the particularity to introduce prior knowledge to the system though a grammar describing the solution structure, there exist several variants of this grammar based genetic programming and these variants are classified under two categories, some belongs the the solution encoding individual representation approach category [19] and the others to the production rule sequence encoding solution representation approach category.

1.1 Motivations

Using Evolutionary Algorithms to automatically generate classification models have been widely used lately, these model are specific for a certain domain and a certain data set, it would be interesting to move forward and try to automatic the design of data mining algorithms and classification ones, these algorithms should not be tailored to a certain domain of course, this will be considered as meta learning.

Another motivation to automatic the design of rule base classifiers is to avoid the human perception and bias. When an algorithm is being manually designed its parameters choice is a very hard task on the designer and this latter will most likely not choose the best ones this is due to the greedy search method used by human. However when this process of designing an algorithm is done automatically we will probably get the best optimal parameters. If go through the literature we find that each algorithm (Rule based classifiers) performs well on some data sets and worse on others, there no algorithm that performs the best on all existing data sets [19,101]. Using algorithms meta-learning should be beneficial and may produce algorithms that performs good on all possible domains.

1.2 Aims and Objectives

The aim of this thesis was to create a new system combining grammatical evolution along with a grammar describing the overall structure of rule based classifiers in order to automatically evolve these latters.

Our second objective was to compare results obtained from our new Grammatical Evolution based system with those produced by the Grammar based GP introduced by Gisele Pappa in [19].

The third objective of this work was to compare automatically evolved rule induction algorithms with the basic manually designed ones, this comparison is lead under the use of several data sets from different domains. The comparison of the evolved algorithms with baseline ones was based on the predictive accuracy of each classifier on each data set.

The fourth objective was to extend the system into a one that can evolve fuzzy rule induction algorithms and eventually we proposed a new grammar that describes the structure of fuzzy rule based classifiers.

1.3 Contribution

This thesis shows that using grammatical evolution together with a grammar defining the structure of rule induction algorithms can produce rule based classifiers that can compete with the manually designed ones. The main contributions of this work are :

- A new system that have the ability to automatically evolve rule based classifiers using grammatical evolution with an already defined grammar [19].
- A grammar, which represents the basic components of fuzzy rule based classifiers
- The use of the new system to generate rule induction algorithms that are able to

produce fuzzy rule sets

1.4 Thesis Organization

The remainder of this thesis is organized as follows.

- Chapter 1 : describes the concepts of standard genetic programming with its different variants. Grammatical evolution was described in a detailed way
- Chapter 2 : presents the basic concepts and methods of data mining and more specifically the classification task. It reviews also the sequential covering strategy followed by algorithms performing rule induction task. This chapter presents an overview of rule evaluation methods and pruning strategies.
- Chapter 3 : describes the proposed system and presents all its components along with its structure. This chapter introduces the grammar used by the system and how possible solution (rule induction algorithms) are evolved.
- Chapter 4 : reports results of a large set of computational experiments using the AGGE system. It reports results obtained by the evolved robust rule induction algorithms (rule based classifiers), which were evolved to be applied to any classification data set. In this chapter we find a comparative study between the proposed system (AGGE) and Gisele's system (GGP).
- Chapter 5 : introduced an extension of the proposed system that was used to automatically evolve fuzzy rule induction algorithms. It also presents the grammar describing the fuzzy set covering paradigm. Different experiments that were conducted using the (FuAGGE) system are reported also in this chapter with a discussion of these results.
- Conclusion : presents the conclusions of the this work and describes future research directions

CHAPITRE 2

Genetic programming

2.1 Introduction

The use of evolutionary algorithms (EA) in artificial life that simulates the natural process of evolution started in the 1960's with works of Alex Fraser and Nill Brecilli, The writings of Jhon Holland on genetic algorithms [1], Ingo Rechenberg on evolutionary strategies [2] and Jhon Koza on genetic programming [3], made this field very popular. Evolutionary algorithms consist of several heuristics, which are able to solve optimization tasks by imitating some aspects of natural evolution.

In real world applications we will most likely encounter problems that are hard to solve such as traveling sales man or knapsack problem, time series prediction or credit scoring, data mining and many more. The dramatic increase in the power of computers allowed the researchers to solve complete complicated real world problems inducing the automatic generation of computer programs which have proven their ability to solve multidimensional problems more efficiently than software produced by human designers. Genetic programming is the method the mostly used among evolutionary heuristics, it is the most popular one specially when dealing with automatic program generation.

In nature evolution is mostly determined by natural selection of different individuals competing for resources in the environment, those individuals that are better are more likely to survive and propagate their genetic materials, the encoding of genetic information is done in a way that admits asexual reproduction which results in off-springs that

are genetically identical to the parents. Sexual reproduction allows some exchange and reordering of chromosomes, producing off-springs that contain a combination of information from each parent. This is the recombination operation which is often referred to as crossover because of the way strands of chromosomes crossover during the exchange. The diversity in the population is achieved by the mutation operator [30], evolutionary algorithms follows almost the same process described above.

In this chapter we will focus on the Genetic Programming method. In 1948 Alan Turing had the first idea of intelligent machinery [31] where he proposed to investigate whether it is possible for machinery to show an intelligent behavior and he proposed three ideas that could serve such purpose and one of these ideas was the evolution.

In 1958, Richard Friedberg described in his work [32] his idea if the possibility of teaching machines to perform certain tasks without teaching them how to do, then in 1959 Arthur Samuel made the first program that performing learning task(a program that was able to play checkers) and in 1963 he used the term Machine Learning to describe the process of computers programming themselves.

In 1985 Michael Cramer gave the first statement of the modern tree based genetic programming, that is procedural languages organized in tree based structures and operated by suitably defined genetic algorithm operators.

In 1992 Jhon Koza greatly expanded Cramer's work and used it for optimization and search problems, and in 1996 Tom Mitchell changed the definition of machine learning into : Computers having the ability to improve through experience. Genetic programming has a population of individuals improving through experience, so the genetic programming is a part of the research field machine learning (see Figure 2.1).

If we go through the literature to see why genetic programming appeared we find that the demand for computer codes kept growing and the process of writing programs was still done manually [33,113] and to keep up with the hardware advances we needed find

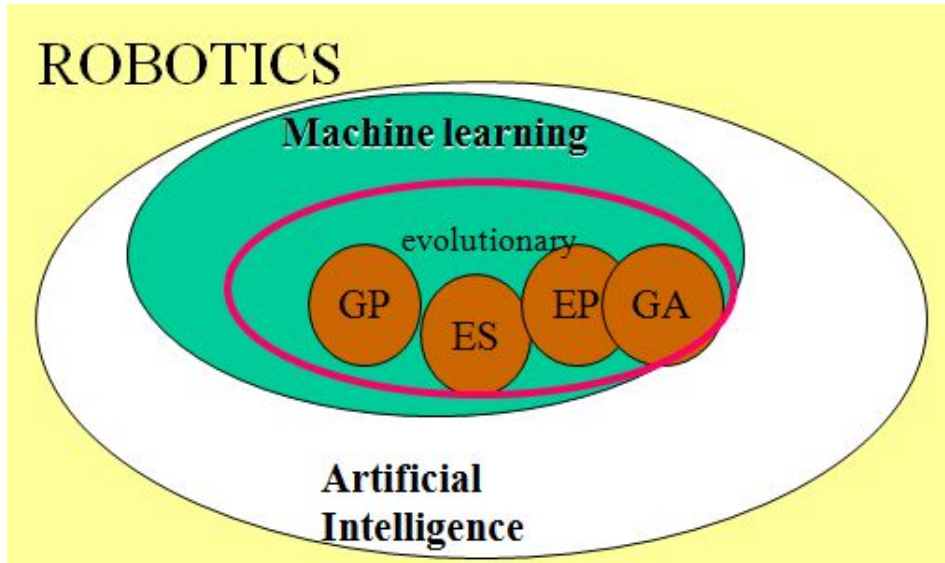


FIGURE 2.1 – Genetic Programming Research Field

a new quick and faster way to write codes, so the idea of automatic programming appeared and genetic programming was the tool to do it because these latter are robust and have an adaptive search method performing a global search in the space of candidate solutions [34]. Genetic programming also copes with attribute interaction and can discover interesting knowledge that other techniques miss. However even though it is the most recent important new model for evolutionary computation and its community is growing rapidly due to its advantages, it has several drawbacks and limitations and therefore some variants of it appeared [30] and each variant serves a certain situation or tackles a certain problem.

GP is an extension of the conventional Genetic Algorithm in which the structures undergoing adaptation are hierarchical computer programs of dynamically varying size and shape. The search space in GP is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. In applying GP there are four major components :

- Terminals and Functions sets used to create candidate solutions
- Initialization method and the individuals representation type which can be (Bit

strings, Real numbers, Permutations of element, Lists of rules, Program element and almost any data structure)

- The fitness function is an objective function that is used to summaries, as a single figure of merit, how close a given design solution is to achieving the set aims, it is used to measure the quality of individuals.
- Selection, mutation and crossover operators used to evolve new off-springs

Selection is the process by which we choose individuals to be processed in some way. Because we have limited processing power, and we would like for GP to solve problems as effectively and efficiently as possible, selection helps us narrow the search space to hopefully optimal bands. We will be using selection to choose which individuals participate in genetic operations in GP[121]. The mutation operator is the ultimate source of genetic variation. Radiation, chemicals change genetic information and they are methods to perform natural mutation, mutation causes new genes to be created in one chromosome it is an asexual and its occurrence is very rare. Mutation is performed by probabilistically selecting a single parental program from the population based on fitness then the mutation point randomly chosen then the sub-tree rooted at that point is deleted, and a new sub-tree is grown there using the same random growth process that was used to generate the initial population. Crossover is performed by selecting two parental programs from the population based on fitness, the crossover point is randomly chosen in the first and second parent. The first parent is called receiving and the second parent is called contributing. The sub-tree rooted at the crossover point of the first parent is deleted and it is replaced by the sub-tree from the second parent. Crossover is the predominant operation in genetic programming (and genetic algorithm) research it is performed with a high probability (say, 85% to 90%).

Genetic programming have also two algorithm types that can be used, these methods are :

- Generational GP : entire populations replaced with each iteration Steady-state
- GP : a few members replaced each generation

Figure 2.2 represents the standard genetic programming pseudo code

```

                                {M=Population Size; N=Maximum No. of Run}
Run: =0;
FOR I: =1 TO N DO
BEGIN
    Gen: =0
    Generate Initial Population Randomly
    WHILE NOT (Terminate Condition for Run)
        BEGIN
            Evaluate fitness of each individual;
            FOR J: =1 TO M DO
                BEGIN
                    Op: =Select Genetic Operator;
                    CASE Op is
                        UNARY:
                            BEGIN
                                Select one individual based on fitness;
                                Perform Reproduction with Probability Pr;
                                Copy the offspring into new Population;
                            END;
                        BINARY:
                            BEGIN
                                Select two individuals based on fitness;
                                Perform Crossover with Probability Pc;
                                J: =J+1;
                                Insert two offspring into new Population;
                            END;
                    END;
                END;
            Gen: =Gen+1;
        END;
    Designate Result for Run;
    Run: =Run+1;
END
END.

```

FIGURE 2.2 – Genetic Programming Pseudo Code

after reviewing the standard GP we will review some of it's variants, if we go through

the literature we find the Linear Genetic Programming (LGP) in which we use a linear genome along with a steady state approach and it uses imperative program languages such as C/C++, the number of registers in this method is an important parameter [35]. Cartesian Genetic Programming (CGP) is also a GP variant that uses an indexed graph (network of nodes) and represents computational structures such as mathematical equations, circuits, computer programs ...etc as strings of integers [36].

Traceless Genetic Programming (TGP) has the particularity to not store evolved programs and uses two operators (insertion and crossover) it is mostly used when the way we find the solution does not matter [37]. The Multi Expression Genetic Programming (MEP) uses a chromosome that represents multiple expressions and the number of genes per chromosome is fixed while the genes length is variable and each chromosome can be seen as a forest of trees [38]. We can also find the (GEP) which is a GP variant called Gene Expression Programming when using this method we use a linear chromosome representing expression trees, the genotypes and phenotypes in this technique are different structurally and functionally. This method uses the generational algorithm, mutation, crossover and the transcription algorithm operators [39].

One of the major problems faced when designing any kind of a GP system is the selection of the terminal and functions sets, These sets should be chosen so as to satisfy the closure property [4].

A set has closure under an operation if performance of that operation on members of the set always produces a member of the same set. For example, the real numbers are closed under subtraction, but the natural numbers are not : 3 and 8 are both natural numbers, but the result of $3 - 8$ is not a natural number. Another example is the set containing only the number zero, which is a closed set under multiplication [121]. The closure property in genetic programming requires that each function is the function set should be able to handle and process the values received as inputs to it, either these

inputs are terminals or outputs of other functions. In order to respect this property the early designed genetic programming algorithms dealt only with only one type of data which reduced of course the ability and the power of the GP system, recent GP systems use new approaches to deal with such problem and at the same time give the GP the ability to deal with different data types, these approaches were designed to satisfy the closure property.

Grammar based genetic programming [5] is the technique the mostly used to tackle the closure property, some of the systems that adopted this strategy are : Context Free Grammar based Genetic Programming (CFG-GP [5], Logic based GENetic PROGRAMMING (LOGENPRO) [6], Grammatical Evolution [7] and the Genetic Algorithm for Developing Software (GADS) [8]). Figure 2.3 illustrates the genetic programming taxonomy

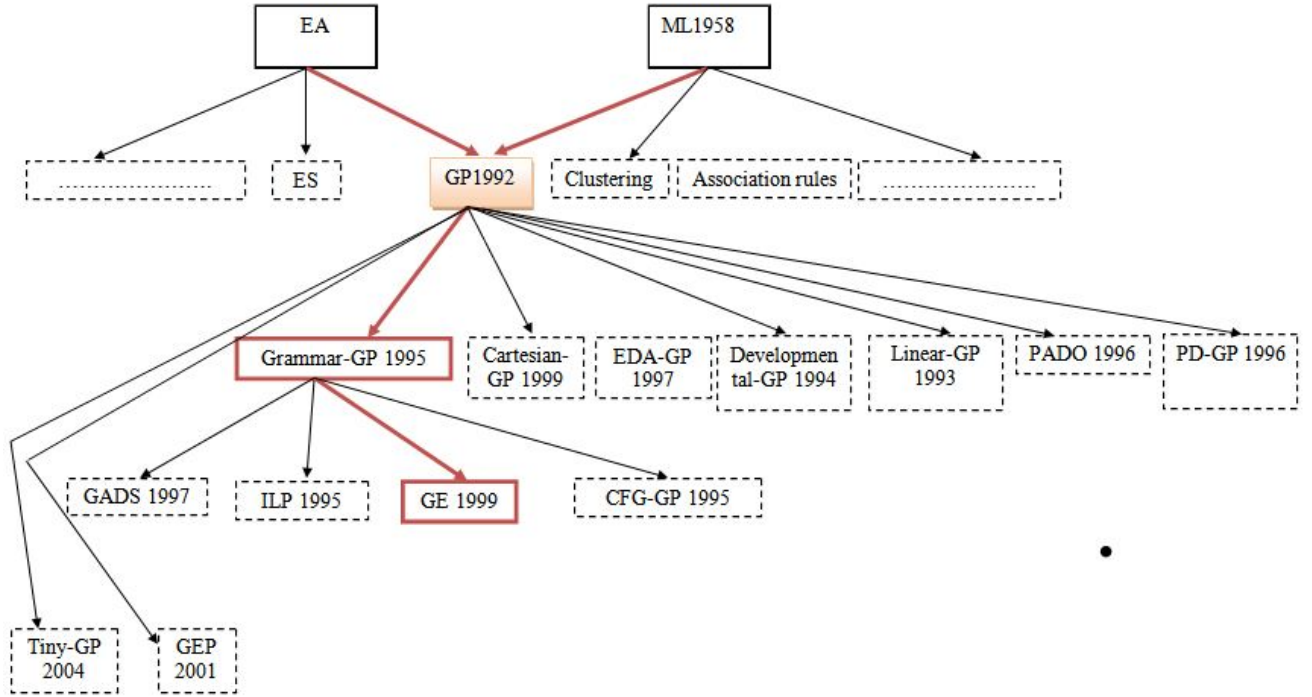


FIGURE 2.3 – Genetic Programming Taxonomy

2.2 Grammar Based Genetic Programming Methods

Writing a computer program is a purely manual task that is time consuming and requires a lot of reflection it is a really tedious task, the idea of automating this process appeared in the 1950's when Arthur Samuel thought of giving computers the ability to learn without being explicitly programmed, this process was called back then Machine Learning, the in the 1980's the Machine Learning definition was changed by Tom Mitchell into "computers having the ability to learn via experience", at the same time another computer science field has appeared which was directly applied to automatic code generation this latter was called Genetic Programming. After several years of research in this new field a new branch of it called Grammar Based GP appeared which was conceived in order to satisfy the closure property and to restrict the search space as well.

2.2.1 Grammars

Block structure of different languages have been used by linguists to describe grammars of these languages and how sentences are built up from smaller units (words and words elements) recursively. Block structure was introduced into computer programming languages by the Algol project (1957-1960), which, as a consequence, also featured a context-free grammar to describe the resulting Algol syntax. This became a standard feature of computer languages, and the notation for grammars used in concrete descriptions of computer languages came to be known as Backus-Naur Form, after two members of the Algol language design committee. The "block structure" aspect that context-free grammars capture is so fundamental to grammar that the terms syntax and grammar are often identified with context-free grammar rules, especially in computer science. Formal constraints not captured by the grammar are then considered to be part of the "semantics" of the language [28]. There exist multiple grammar types such as : context free grammars, logic grammars, affix grammars, attribute grammars, indexed grammar ... etc.

Context Free Grammars

A context-free-grammar (CFG) G is a quadruple (V, L, R, S) where :

V : a set of non-terminal symbols

L : a set of terminals

R : a set of rules

S : a start symbol

Example :

$V = q, f,$

$$L = 0, 1$$

$$R = q \longrightarrow 11q, q \longrightarrow 00f, f \longrightarrow 11f, f \longrightarrow \xi$$

$$S = q$$

Logical Grammars

Logical grammar symbols (terminal or non-terminal) include arguments that can be any term in the grammar, this is the main difference between context free grammars and the former ones.

Terms in logical grammars can be either logical variables represented by a question mark "?" and followed by a string of letters or digits, functions which are grammar symbols followed by a bracketed n-tuple of terms or 0-arity functions called constants, these arguments are used to enforce context-dependency and to construct tree structures in the course of parsing, such tree structures can provide a representation of the semantics (meaning) of the program.

- Terminals : which are enclosed in square brackets, correspond to the set of words of the language specified.
- Non-terminals : are similar to literals in Prolog, commas denote concatenation and each grammar rule ends with a full stop.
- Right-hand side of a grammar rule may contain logic goals and grammar symbols. The goals are pure logical predicates for which logical definitions have been given. They specify the conditions that must be satisfied before the rule can be applied [6,108].

In the next section we will see some of the Grammar Based Genetic Programming methods found in the literature.

2.2.2 Genetic Algorithm for Developing Software (GADS)

Genetic Algorithm for Developing Software as described by Patterson and Ia [9] is a technique for genetic programming where the genotype is different from the phenotype, The syntax phenotype language is described using a BNF as a set of productions of the form Left Hand Side :=Right Hand Side Where the left hand side is a Non-Terminal symbol and the right hand side can be a Terminal symbol as well as a Non-Terminal one.

In GADS the BNF productions are numbered from 0 to n and each gene is generated as a random number s, having s in $[0..n]$, and of course the chromosomes encoding rules to be applied are of a fixed length. The phenotypes are generated either as strings or abstract syntax trees using the encoded rules in the chromosome, using this schema we can face the problem of inapplicable production rules when interpreting a gene, Paterson [9] suggested that this kind of production rules will be ignored and he stated that if at the end of the chromosome we find a gap i.e. a Non-Terminal which didn't have any terminal chosen for its replacement, then a default value is inserted [6]. The main elements of the GADS system are :

- The GA Engine which maintain a population of arrays of integers called chromosomes .
- The Ontogenic Mapper that maps and converts the chromosomes into phenotypes (program codes).
- The Fitness Evaluator which measures the quality of the program fragments (phenotypes).

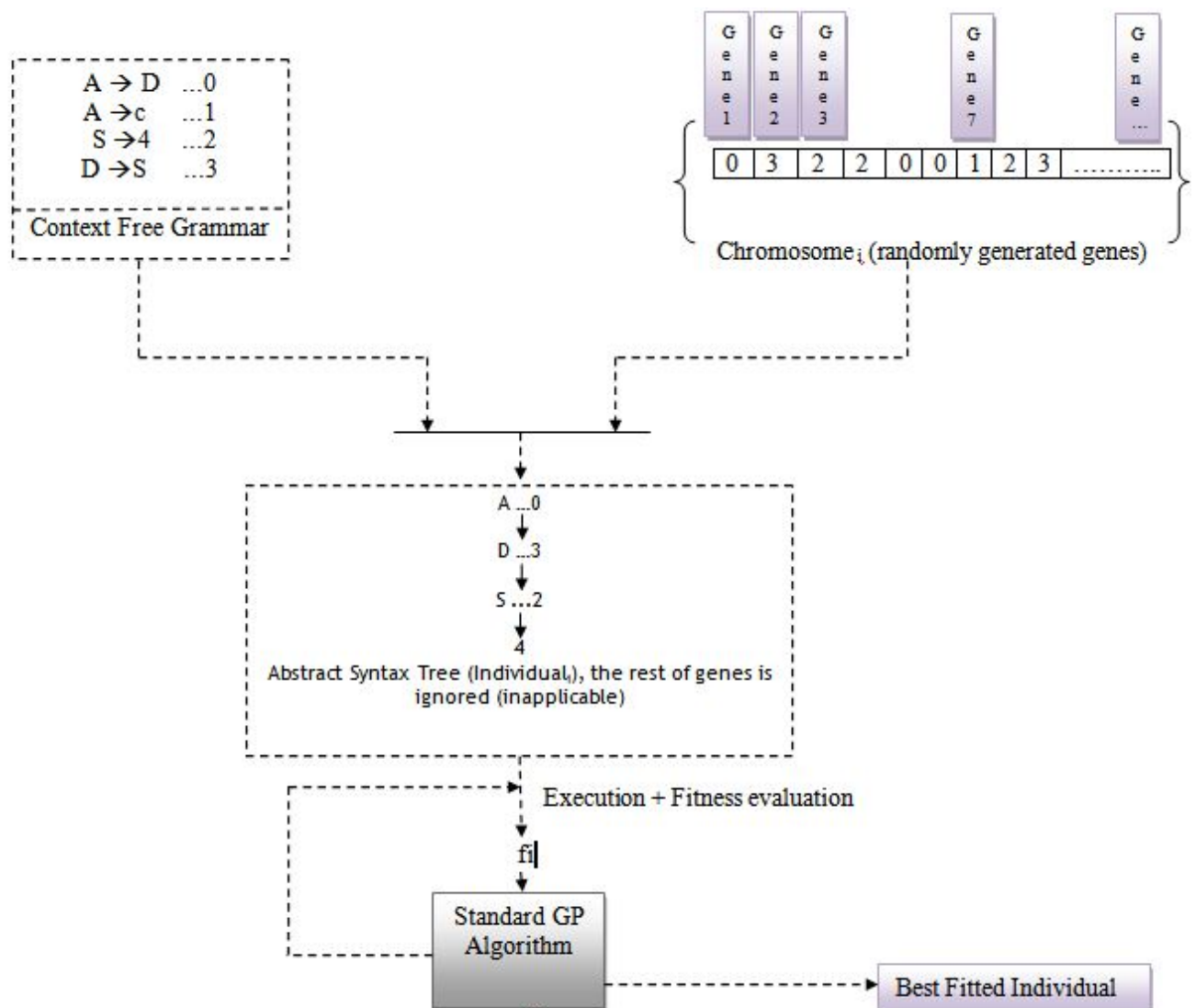


FIGURE 2.4 – Genetic Algorithm for Developing Software (GADS) Process

2.2.3 Logic Grammar Based Genetic Programming (LOGEN-PRO)

Wong and Ia[6] presented a new genetic programming technique and framework based on logic grammars called (LOgic grammar based GENetic PROgramming system) LOGENPRO the idea of this system was to combine genetic programming with the ILP to induce knowledge from databases [10], this system have the same main components as a tree based GP [3], but it uses inductive logic grammars in addition to it, this framework was implemented in Lisp programming language and it emulates logical programming to interpret the grammar. The grammar employed by LOGENPRO is described using a notation similar to that of definite clause, the logical goals in LOGENPRO are procedures written in Lisp and not only logical goals as in logical programming. Individual representation is conceptually the same as in the CFG-GP the only difference is that the derivation trees have structured nodes containing terms and goals. The phenotype is generated as an AST where each node of these ASTs is a 3 integers-tuple component : (Symbol L1 L2). The operators in LOGENPRO are not simple and are modified so they can respect the grammar when applied on the individuals of a certain population, the sub trees containing logical goals do not undergo these operators because they are considered as atomic immutable components

2.2.4 Context Free Grammar based Genetic Programming (CFG-GP)

The use of formal grammars was firstly introduced by Peter Whigham in order to control the search algorithm of genetic programming, it was also a solution provided to solve the typing problem recognized by Montana [11] and a mean to introduce more bias into the genetic programming. Peters Whigham proposed a method called Context Free

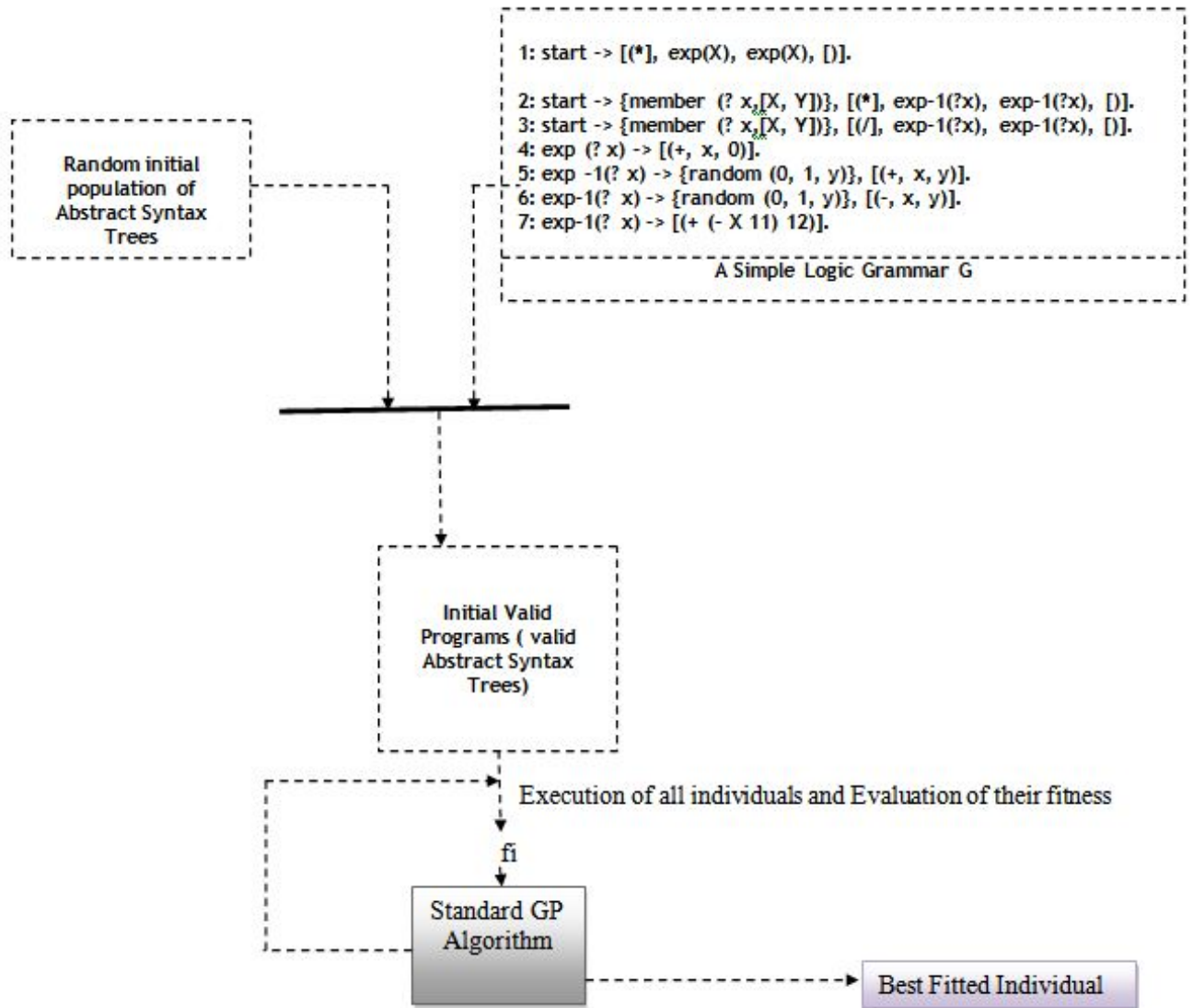


FIGURE 2.5 – Logic Grammar Based Genetic Programming (LOGENPRO) Process

Grammar GP [12,13] and noted that the use of CFG can be in a similar way to that of typing that restricts the structure of candidate solutions, this new method is based on the redefinition of the elements of tree based GP so it respects a certain grammar G . Individuals in CFG-GP have the tree structure and they are derived according to the CFG introduced to the GP, the genetic operators are modified in way to preserve this representation.

Context Free Grammars allow to easily use programming languages most appropriately for a given problem [14]. The learning bias mechanism proposed by Whigham [5] is as follows :

- Let each production rule in the grammar G have an integer weight called "fitness".
- In every generation find the fittest individual.
 - Choose one of the deepest Non-Terminals b in its derivation tree.
 - Let a be the string of terminals used to rewrite b .
 - If b is a singleton the let A be the highest node in the chaine [14].
 - Else let $A=b$.
 - Create a new rule $A \rightarrow a$ if it's already in the grammar then increase its weight by 1.
 - Otherwise add $A \rightarrow a$ with a weight equal to 1 to the grammar G
- When applying mutation or replacement it selects a production rule with a probability proportional to their weight.

2.2.5 Grammatical Evolution (GE)

Grammatical evolution [15] a special case of grammar based genetic programming, it has as an objective global optimization, it also can be considered as a technique for automatic programming, it is related to other evolutionary algorithms for evolving programs such as genetic programming, gene expression programming as well as genetic algorithms

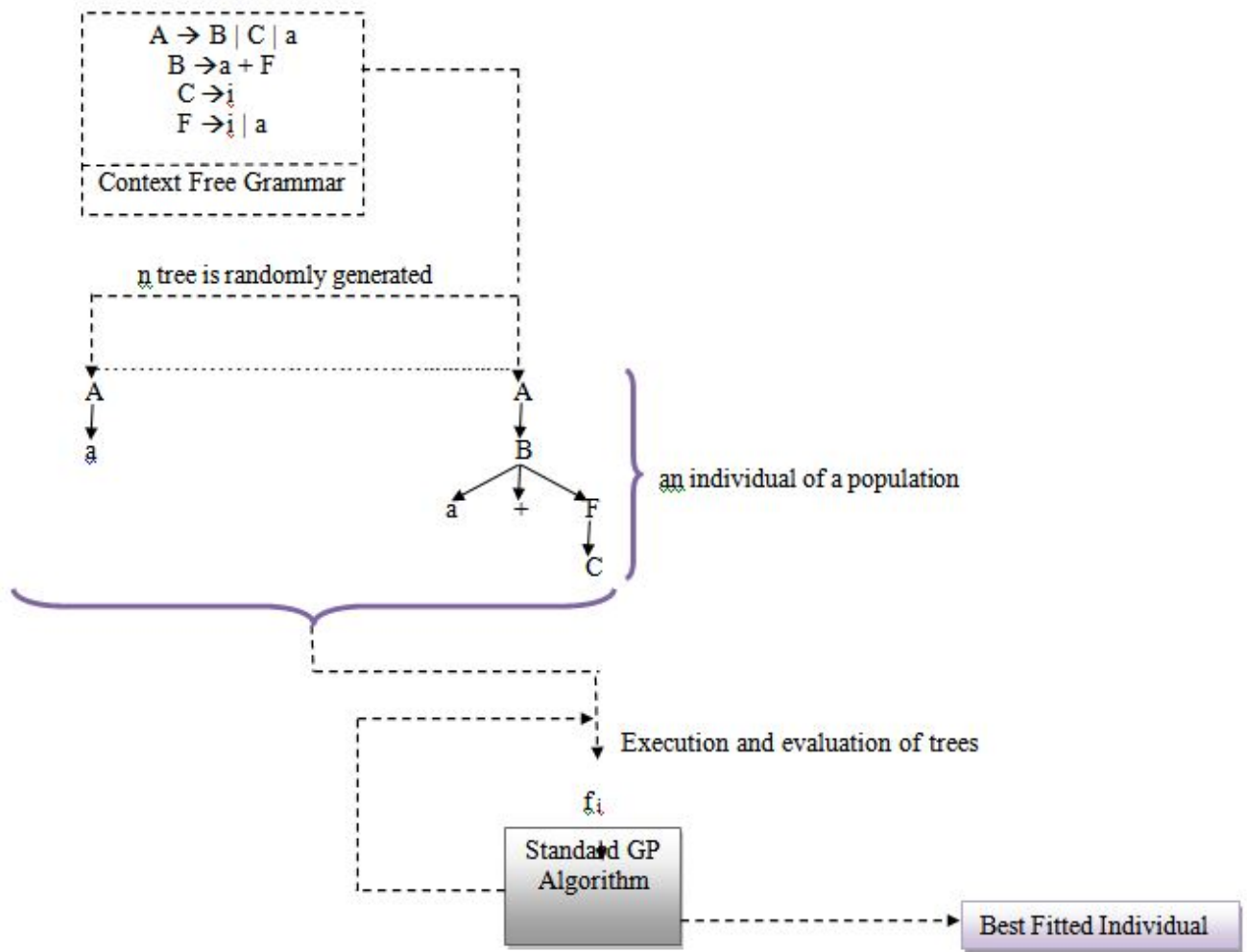


FIGURE 2.6 – Context Free Grammar based Genetic Programming (CFG-GP) Process

that uses binary strings [40].

Grammatical evolution is a bio-inspired method, its principle was borrowed from the natural protein generation using genetic materials, the genome is comprised of DNA strings which are series of building blocks willing to be transcribed to RNA codons, then this RNA codons will be translated into amino acids and used in a protein, this protein is the so called Phenotype. In the artificial evolution process the phenotype is an actual computer program and the Genotype is a binary string representing the former, this binary string is decoded into a sequence of integers that will in turn be mapped

into rules making up the program, this process is called mapping.

Grammatical evolution method needs a grammar as input, this grammar is defined using the Backus Naur Form (BNF), which is a context free grammar representation as stated in Section 2.2.1, this grammar contains a list of rules. Individuals are represented as binary variable length strings, these strings are divided into codons of 8 bit sequences. The codons are decoded into integers in the range $[0 - 2^8 - 1]$ and if the binary string is reached when reading integers, the reading process loops back to the start of the string [25]. The integers are mapped to expressions from the BNF grammar until a complete syntactically correct expression is achieved or formed. When mapping a certain genome to a computer program we may not use all genetic materials or we may use it more than once given its circular nature[27]. Figure 2.7 illustrates the mapping process of grammatical evolution in natural and artificial life.

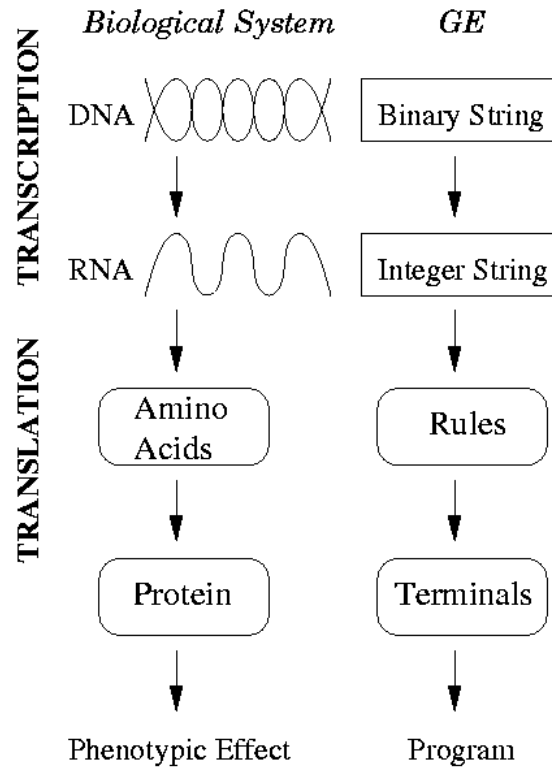


FIGURE 2.7 – Grammatical Evolution Mapping Process

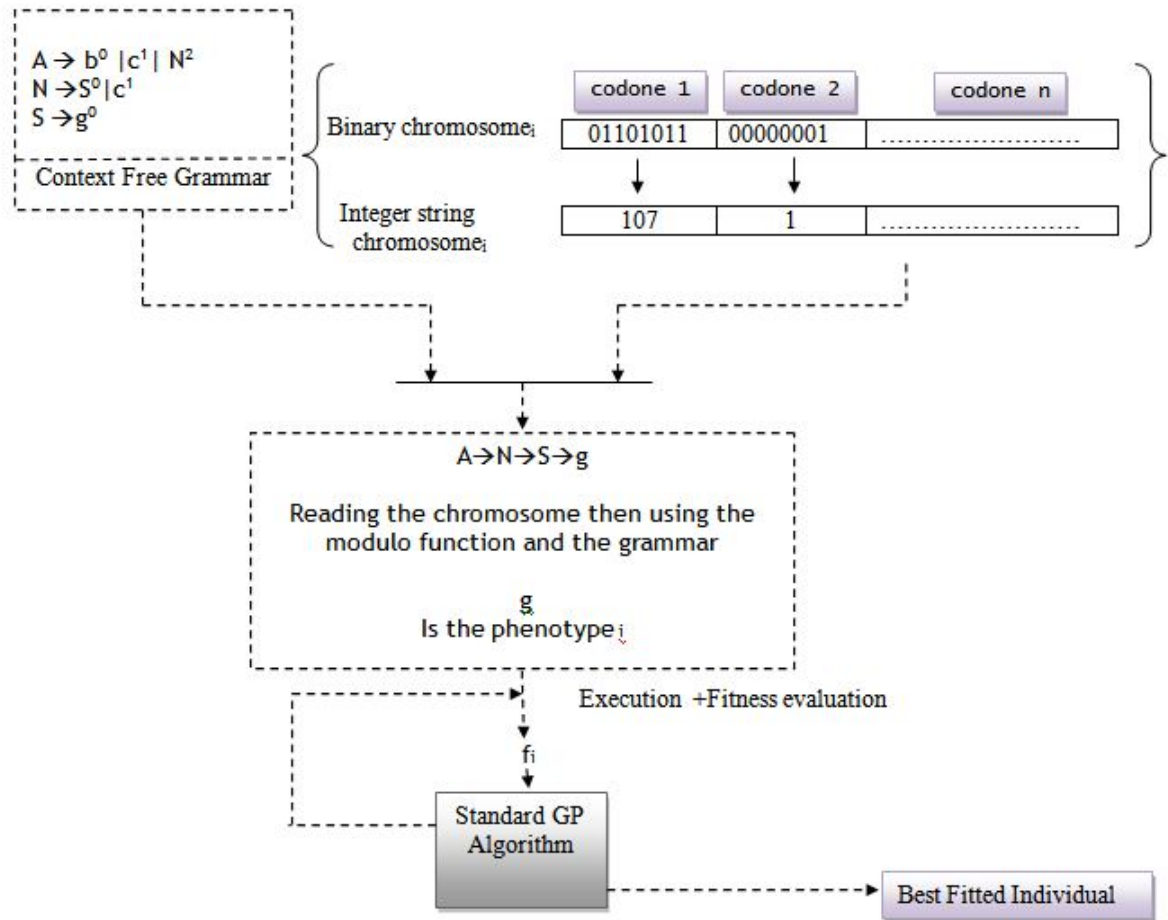


FIGURE 2.8 – Grammatical Evolution (GE) Process

After the mapping process is done, the resulting program will be execute and its fitness is determined so it can be able to compete with the remaining individuals of the population. Figure 2.8 shows the grammatical evolution process.

2.3 summary

This chapter introduced the concept of standard genetic programming and grammar based genetic methods (GGP). The problem of closure is fixed with the use of grammars along with the GP, grammars are also a way to introduce prior knowledge about he

solution to bias and guide the search.

This chapter also described different variants of grammar based genetic with an programming methods along with an overview of types of grammars commonly used with GP. Grammatical evolution was presented in a more detailed way because it will be used in the next chapters and it is the core of the proposed system.

CHAPITRE 3

Data Mining

3.1 Introduction

Information stocked nowadays in databases can range in size into terabytes, this is due to the huge amount of data collected and warehoused, data coming from web, e-commerce, grocery stores, banks and credit card transactions...etc.

Collecting and warehousing data have become easy due to the availability of cheaper and very powerful computers. Storing data can help in providing better, customized services for an edge and this can be achieved by analyzing this data that is stored at enormous speed (ex : remote sensors on a satellite, telescopes scanning skies, micro-arrays generating gene expression data...etc).

Traditional approaches for deriving knowledge from data rely strongly on manual analysis and interpretation. For any domain : scientific, marketing, finance, health, business, etc. the success of a traditional analysis depends on the capabilities of one/more specialists to read into the data : scientists go through remote images of planets and asteroids to mark interest objects, such as impact craters ; bank analysts go through credit applications to determine which are prone to end in defaults. Such an approach is slow, expensive and with limited results, relying strongly on experience, state of mind and specialist know-how. Moreover, the volume of generated data is increasing dramatically, which makes traditional approaches impractical in most domains. Within the large volumes of data lay hidden strategic pieces of information for fields such as science, health

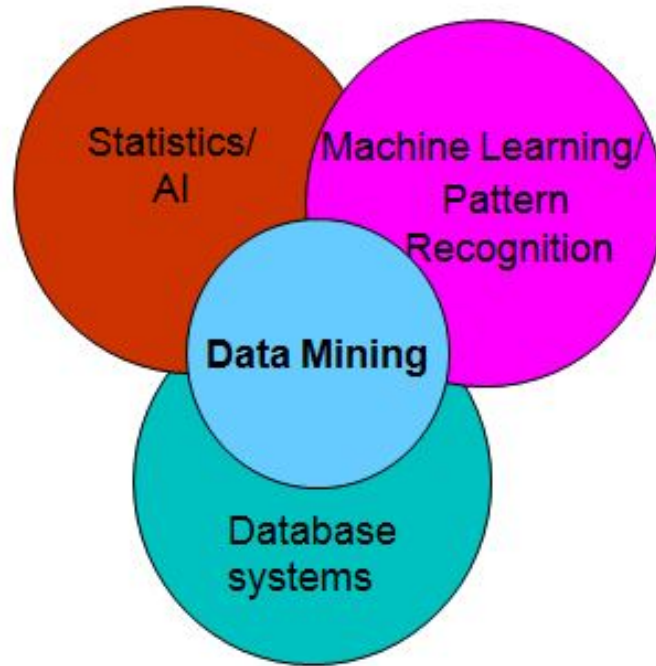


FIGURE 3.1 – Data Mining Field Origins

or business. Besides the possibility to collect and store large volumes of data, the information era has also provided us with an increased computational power. The natural attitude is to employ this power to automate the process of discovering interesting models and patterns in the raw data. Thus, the purpose of the knowledge discovery methods is to provide solutions to one of the problems triggered by the information era : "data overload" [56]. In 1990's the data mining field have arisen, even though its roots are tracked back along three family lines : Statistics, Artificial Intelligence and Machine Learning. Figure 3.1 shows the origins of the data mining field.

Using data mining researches are now able to classify and segment new data in an easy way.

3.2 Data Mining and Analysis

In the masses of data stored there is often hidden information that is not readily evident and human analysts may take weeks and months to discover useful information and much of data is never analyzed at all, which could be considered as a waste of possibly interesting information. We can classify all data mining techniques under two task types :

- Prediction methods : where we use variables to predicts unknown or future values of another variable, tasks following this method are : Classification, Regression and Deviation detection.
- Description methods : where we try to find human interpretable patterns that describe data. Tasks following this strategy are : Clustering, Association rule discovery and sequential pattern discovery.

When dealing with classification we try to find a model for the class attribute as a function of the values of other attributes, given a collection of records where each record contains a set of attributes and one of the attributes is the class. After finding the model we will assign a class as accurately as possible to a previously unseen records, to determine the accuracy of the model we use a test set. Classification can be used to predict fraudulent cases in credit card transaction.

Clustering goal is to find clusters, given a set of data points, each having a set of attributes and a similarity measure among them. The clusters we try to find should respect two points, firstly data points in one cluster are more similar to one another and secondly data point in different clusters are less similar to one another. Euclidean distance can be used a similarity measure when the attributes are continuous. Clustering can be used to subdivide a market into distinct subsets of customer where any subset may conceivably be selected as a market target to be reached.

In association rule we have a set of records each of which contains some number of items from a given collections and we try to find and produce dependency rules which will

predicts occurrences of others items.

The aim of the sequential pattern discovery is to find rules that predict strong sequential dependencies among different events given a set of objects where each object is associated with its own time line. Rules are formed by first discovering patterns, Event occurrences in the patterns are governed by timing constraints.

Regression is widely used in data mining and it is used to predict a value of a given continuous valued variable, based on the values of other variables assuming a linear or non linear model of dependency, it is a method that is greatly studied in statistics. Regression can be use to predict wind velocities or to predict sales amounts of a new product.

In order to detect significant deviations from normal behavior such are credit card frauds we use the deviation detection model. After reviewing different data mining tasks, we will focus on the classification one and we will see different techniques to perform this task in the following section.

3.3 Classification in Data Mining

Classification goal is to be able to classify previously unseen records using a model that was generated using a collection of records as stated before. Classification can be used to predict for example the type of tumor cells as being benign or malignant, it also be used to categorize new stories as finance, weather, entertainment, sport, etc. There exist several classification techniques such as :

- Support Vector Machines
- Naive Bayes and Bayesian Belief Networks
- Neural Networks
- Memory Based Reasoning

- Decision Tree Based Methods
- Rule Based Methods

When using support vector machines [63] we try to find a linear hyper plane (decision boundary) that will separate the data and at the same time maximize the margin between the closest points to the hyper plane, from different classes .

Neural Networks [64], are an assembly of interconnected nodes and weighted links the output node sums up each of its input values according to the weights of its links. The output value is then compared against a certain threshold to decide to which class the current record belongs.

Bayes classification [65] is a probabilistic framework for solving classification problems, this framework uses the Bayes theorem that considers each attribute and class label as random variables and it tries to predict a class of a certain record by maximizing a conditional probability.

Memory based reasoning [66] tries to predict unknown values for a case based on similarity with k most similar cases, it s reasoning by analogy.

When using Decision tree based methods such as CART [67], C4.5 [68], ID3 [69], etc we try to build a decision tree from a training set. This model have a structure of a tree where leaves represent class labels and branches represent conjunctions of features leading to those class labels.

Now we will see the rule based method in a more detailed way. Rule based classification to classify records by using a set of "IF - THEN " rules, rules of the type (condition) $\rightarrow$ C where condition is a conjunction of attributes and C is the class label, these rule have two characteristics which are :

- Rule coverage : the fraction of records that satisfy the condition part of this rule.
- Rule accuracy : the fraction of records satisfying both condition part and the class one.

The rule set generate as a classification model can be either an ordered set known as

decision list where rules are rank ordered according to their priority or it can be presented as a set of rules where the order does not matter. To build a rule based classification model we have three possible ways to follow, we can follow the indirect way, it is called indirect because rules are extracted from other classification models or knowledge representations such as decision trees [50,51]. C4.5rules is a algorithm following this so called indirect method.

Direct methods is another way used to construct a rule based classifier in which we extract rules directly from the data we have, algorithms following this methods are : RIPPER[60], CN2[80], PRISM[20], 1RULE[122],...etc. All algorithms following this method, follow a strategy called Sequential Covering, this latter will be overviewed in the next subsection and it will be used in the construction of our proposed system.

We can also use evolutionary algorithms, more specifically the genetic algorithm or the genetic programming method. When using genetic programming to produce a rule based classification model we have four steps to follow, first we should choose a representation of the individuals, then we should specify the search method to follow, after that we choose how we will evaluate our generated rule set (fitness function), then we decide how to prune the rule set. In the GP each individual (tree) is a set of rules for the class C and the GP will run i times (i is the number of classes in the data set). However when we use GA we have a choice to make when designing the individuals representation, we either follow the Michigan method where each individual represents a single rule or we follow the Pittsburgh method where each individual represent a set of rules [16]. When only the rule base is being induced, the membership functions are predefined either by a human expert or some other process (e.g clustering), and remain fixed throughout the inductive process. If Michigan-style encoding is adopted, then the EA generally evolves individuals that represent either a rule antecedent or an entire rule. In the first case, at each iteration a separate deterministic procedure is used to determine

the rule consequent before evaluating the rule for fitness, e.g. [116]. Alternatively, the rule consequent, i.e. class, may already be specified for each rule antecedent during the entire run of EA. An EA is run several times in succession with each run concentrating on evolving rule antecedents pertaining to a specific class, e.g. [115 ; 117]. If a complete rule is encoded in an individual, then the rule consequent may also be subject to evolution and change by the genetic operators. A restriction may be placed on the crossover operator to ensure that only parents belonging to the same rule consequent are combined to produce offspring [118 ; 119]. With a Pittsburgh-style approach, generally both rule antecedents and associated consequents are encoded. The genetic operators may then act on the individual rules within a rule base, or on the composition of the rule base itself. An alternative is proposed in [120], where each individual represents a fixed number of rule antecedents. The consequents of the rules are dependent on the number of positive and negative examples they match in the training set and are determined prior to evaluation of the rule base.

3.4 Sequential Covering

Sequential covering strategy have four main steps to respect, first we start from an empty rule then we grow a rule using a function called Learn-One-Rule, the we remove training records covered by this rule, these two last steps are repeated until a stopping criterion is met. The aspects of sequential covering are :

- Rule Growing
- Instance Elimination
- Rule Evaluation
- Stopping Criterion
- Rule Pruning

Figure 3.2 represents the sequential covering pseudo code and Figure 3.3 represents the Learn-One-Rule function pseudo code

```
SequentialCovering(target_att,atts,examples,threshold)

Learn_rule ={};
Rule=Learn_One_Rule(target_att, atts ,examples);
While ( performace(rule, examples) > threshold ) do
    Learned_rules=learned-rules +rule ;
    examples= example - examples correctly classified by rule ;
    rule= Learn_One_Rule( target_att,atts,examples);
Done
Learned_rules = sort(Learned_rules(performance));
Return Learned_rules ;
```

FIGURE 3.2 – The Sequential Covering Pseudo Code

```
Learn_One_Rule(target_att, atts ,examples)

P=positive_examples;
N=negative_examples;
While (p) do
    learning a new rule
    new_rule= most general rule
    new_rule_n= n
    While (new_rule_n) do
        add new literals to specialize new_rule
        Condidate_literals =generate candidates
        Best_literal = argmax (performance(specialize_rule(new_rule , L)),L in
        Add Best_literal to new_rule preconditions;
        new_rule_n= subset of new_rule_n satisfing new_rule preconditions;
    Done
    Learned_rules=Learned_rules+new_rule;
    p=p-{members of p covered by new_rule} ;
Done
Return Learned_rules;
```

FIGURE 3.3 – The Learn One Rule Pseudo Code

The CN2[80] algorithm starts from an empty rule (an empty rule is a rule with empty

conjunctions) and then it adds conjuncts that minimizes the entropy of the current rule being grown and it determines the rule consequent algorithm by choosing the majority class of instances covered by the rule, while the RIPPER[60] is different it chooses a class then creates a rule with no antecedent called an empty rule, then it adds conjuncts that maximizes FOIL's[50] information gain.

After the rule growing step sequential covering algorithms eliminates instances covered by the accurate rule or the next grown rule will be identical to its previous one, the training set will shrink each time a new rule is added to the model being constructed. To be able to choose a rule among a set of rules we need to know how much is the goodness of each rule, we can measure the goodness of a rule using different methods or metrics, the most trivial known one is the accuracy metric.

3.4.1 Evaluation Metrics

Choosing rules to be added to the classification model depends on the goodness of these rules and to know how much is good a certain rule or which rule is better for the classification purpose, there exist several techniques called evaluation metrics, if we change the evaluation metric used to build a certain model, the model will also change. This section describes heuristics used to estimate rules quality.

A rule quality and goodness can be estimated by a metric called Laplace estimation and this heuristic is the famous one among other metrics. The CN2[80] and the BEXA[52] algorithms use the Laplace estimation to evaluate rules, Equation 3.1 represents the Laplace definition, where p is the number of positive records covered by the current rule, n is the number of negative records covered by the current rule, nC is the number of classes of the data set being used and $rule_i$ is the current rule being evaluated

$$Laplace(rule_i) = \frac{p + 1}{p + n + nC} \quad (3.1)$$

A more simpler evaluation metric is the precision one, it is also called the Confidence and this one is used by SWAP-1[53], IREP[54] algorithms. Confidence is defined in the Equation 3.2 where p is the number of positive records covered by the current rule, n is the number of negative records covered by the current rule and $rule_i$ is the current rule being evaluated.

$$Confidence(rule_i) = \frac{p}{p + n} \quad (3.2)$$

When using this metric rules with higher confidence and low statistics support are chosen over rule with higher statistical support and slightly lower confidence value and this is undesirable, this problem was tackled by using the Laplace estimation.

If we through the most famous and the most used rule induction algorithms following sequential covering strategy such as RIPPER[60] and PRISM[20] we find a new evaluation metric called the information gain, its equation is presented in Equation 3.3, where $rule_i$ is the current rule being evaluated and $rule_{i\alpha}$ is a specialization or generalization of $rule_i$.

$$InfoGain(rule_i) = -\log(Confidence(rule_i)) - \log(Confidence(rule_{i\alpha})) \quad (3.3)$$

There exist also another metric which is a Laplace generalization, when using this metric we use the apriori probability of classes when a rule with no coverage is used. Equation 3.4 describes the m-estimate definition.

$$m - Estimate(rule_i) = \frac{p + (m * \frac{P}{PN})}{p + n + m} \quad (3.4)$$

m is the number of virtual examples added to the training set and that are distributed given the prior probability of classes, P is the total number of positive records of the training set and N is the total number of negative records of the training set.

The accuracy of a rule can be also used to evaluate a rule by dividing the number of correctly classified records over the total number of covered records by the rule being evaluated. Equation 3.5 describes the accuracy definition where p_c is the number of correctly classifier records

$$Accuracy(rule_i) = \frac{p_c}{P} \quad (3.5)$$

3.4.2 Rule Pruning

When building a rule based classification model we look for a model or a set that is complete, which means that it should cover all records in the training sets, it should be also consistent, which means that it should not cover any negative examples, in order to find rules and models that satisfy these two requirements pruning was introduced to rule induction algorithms.

Pruning is used in rule based classification to handle the problem of overfitting as well as noisy data problem and there exist two categories of pruning : the prepruning one which deals with the overfitting the noisy data problem by stopping the refinement process of rules before becoming too specific. when stopping the refinement process we allow a rule to cover few negative examples and when we stop adding rules to the classification model we allow this later to leave some uncovered examples.

The second category of pruning is the postpruning one which deals with rejection and conflict problems and it allows to find a complete consistent model. The aim of postpruning is to improve the learned model and as its name implies it is applied after the mode building stage is finished. When applying postpruning we either remove rules from the model while preserving and improving the modes accuracy or we remove rule conditions. If we go through the literature we find examples of algorithms using the two categories

of pruning. FOSSIL[58], FOIL[50], m-FOIL[57] use the prepruning technique while the REP[59], GROW[60] use postpruning techniques. Figure 3.4 represents a pre-pruning algorithm and Figure 3.5 represents a post-pruning algorithm

```

procedure PrePruning (Examples)

Program = EmptySet
while POSITIVE (Examples) NotEqualTo EmptySet
  Clause = EmptySet
  Cover = Examples
  while NEGATIVE (Cover) NotEqualTo EmptySet
    NewClause = Clause U FIND LITERAL(Clause, Cover)
    if STOPPING CRITERION (Program, Newclause, Cover)
      exit while
    Clause = NewClause
    Cover = COVER(Clause, Cover)
  Examples = Examples â Cover
  Program = Program U Clause
return (Program)

```

FIGURE 3.4 – A Prepruning Algorithm Pseudo Code

```

procedure PostPruning (Examples, SplitRatio)

SplitExamples (SplitRatio, Examples, Growing Set, Pruning Set )
Program = SeparateAndConquer(GrowingSet)
loop
  NewProgram = SimplifyProgram (Program, Pruning Set)
  if ACCURACY( NewProgram, PruningSet) < ACCURACY(Program, PruningSet)
    exit loop
  Program = NewProgram

return (Program)

```

FIGURE 3.5 – A Postpruning Algorithm Pseudo Code

Recently new approaches appeared in the literature such as MDL-GROW[60], TDP[61], IREP[54]. These new approaches combine and integrate the two pruning methods[55]

discussed previously, Figure 3.6 describes an example of the new approach (TDP) that combines prepruning and postpruning together.

```

procedure TDP(Examples, SplitRatio)

Cutoff = 1.0
BestProgram = EmptySet
BestAccuracy = 0.0
SPLITEXAMPLES(SplitRatio, Examples, GrowingSet, PruningSet)
repeat
    NewProgram = FOSSIL (GrowingSet, Cutoff)
    NewAccuracy = ACCURACY (NewProgram, PruningSet)
    if NewAccuracy > BestAccuracy
        BestProgram = NewProgram
        BestAccuracy = NewAccuracy
    LowerBound = BestAccuracy - STANDARD ERROR (BestAccuracy, PruningSet)
    Cutoff = MAXIMUM PRUNED CORRELATION (NewProgram)
until (NewAccuracy < LowerBound) or (Cutoff = 0.0)

loop
    NewProgram = SIMPLIFYPROGRAM (Program, PruningSet)
    if ACCURACY (NewProgram, PruningSet) < ACCURACY (Program, PruningSet)
        exit loop
    Program = NewProgram

return (Program)

```

FIGURE 3.6 – TDP Pruning Method Pseudo Code

3.5 Summary

Data mining is a fashionable research field nowadays and it is a large one too. This chapter presented the basic concepts and methods of data mining and more specifically the classification task, it presented different ways and methods used to build classification models.

In our thesis we are interested in the rule induction algorithms following the sequential covering paradigm. We presented in a detailed way the sequential covering concepts and we presented also an overview of evaluation methods and pruning categories. this will be very helpful when introducing our proposed system.

CHAPITRE 4

Proposed System : Automatic Generator of Rule Base Classifiers (AGGE) and Results

4.1 Introduction

Previous chapters (2-3) of this thesis introduced the two main fields of data mining and genetic programming and their basic concepts. Chapter 2 reviewed the grammatical evolution method in a detailed way and chapter 3 focused on the sequential covering paradigm to induce decision rules. This chapter will introduce a new system called AGGE : Automatic Generation of rule based classifiers using Grammatical Evolution. As it will be shown later, this system will be able to produce and evolve rule based classifiers that are competitive with the baseline well known manually designed one, these evolved algorithms are not tailored to a specific data set or domain and they can be applied and used with any data set. This system (AGGE) will be compared to the GGP one which perform the same task using solution encoding individual approach[19], while our system is based on production rule sequence encoding approach. There is no evidence in favor of any of these two approaches with respect to their effectiveness and in matter of easiness of implementation we can say that the first one lacks the genotype-phenotype mapping process so we do not need to worry about the mapping. However, when using the second one we will be applying genetic operators on binary strings so we do not need to worry about the compliance between the genetic operators and the

grammar. Figure 4.1 illustrates the structure of the AGGE system. We should note that the grammar used in the work was previously introduced by Gisele in [19], this grammar was dedicated to present sequential covering components.

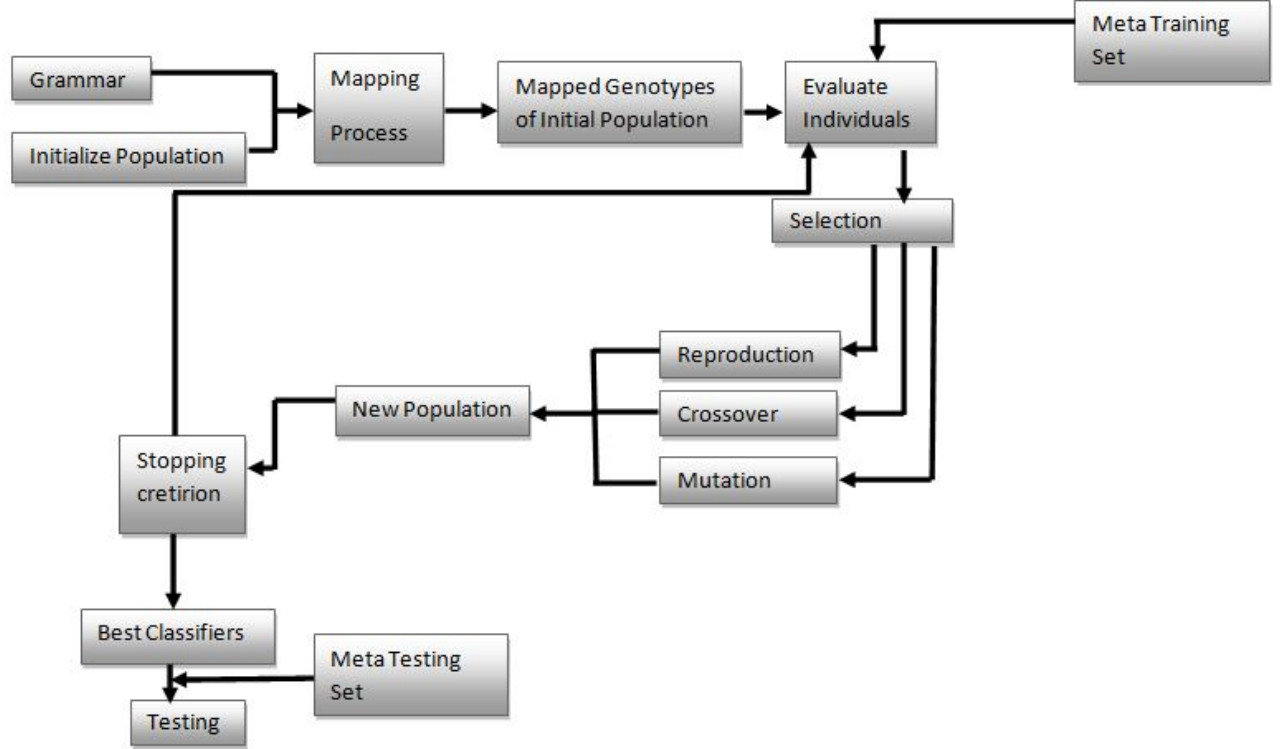


FIGURE 4.1 – AGGE System Scheme

Each individual in any AGGE population is a complete rule based classifier, a classifier that contains all necessary components that a sequential covering algorithm needs, these classifiers have practically the same structure as the baseline classifiers such as : Prism, OneRule, CN2...etc. This grammatical evolution based system will be able to produce individuals respecting the grammar describing sequential covering algorithms then we will evaluate each of this possible solutions using a fitness function and a meta training test, then these individuals will be compared, selected, crossed over, mutated and reproduced until the evolution process meets a stopping criterion and the best rule induction algorithm is fixed.

4.2 Individual Representation

The approach used in our system is different from that used by Gisele in [19], it uses a mapping between the genotype (individual) and the phenotype. The individuals in our system are represented as a linear genome (array of integers) it can also be represented as a binary string. the initial population (genotypes) are generated independently from the grammar and during the evaluation of each individual the system applies a genotype/phenotype mapping process and this is performed using the genetic materials (arrays of integers) to select which rule to choose from the grammar. Figure 4.2 shows how individuals in a certain population are represented. We should note that each individual (genotype) represented as an array of integers represents a whole complete rule induction algorithm.

4.3 Population Initialization

In general a population P is understood as a finite of objects which can be enumerated [70]. The AGGE system uses a ramped half and half initialization method to initialize the population, the population is divided (ramped) between minimum depth (shortest possible depth of a derivation tree) and maximum depth (maximum allowed depth of derivation tree) for each individual .

The population generation procedure starts by calculating the minimum tree depth for all the production rules of the grammar. If we have a rule which generates only terminals we have a minimum depth equal to 1 and if a rule have two or more non terminals having the maximum depth between all the non terminal rules, the minimum depth is calculated in a recursive way.

When we initialize individuals as strings of random numbers, when mapping these in-

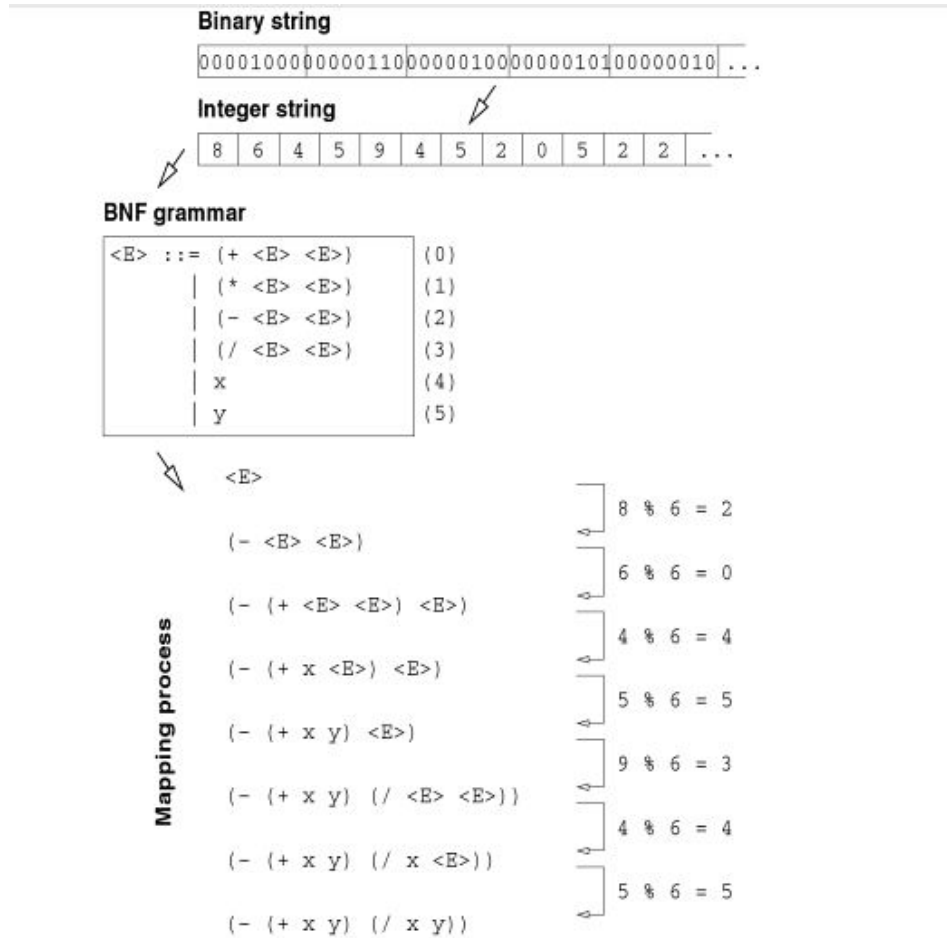


FIGURE 4.2 – Individual Representation

dividuals we have no guarantee that they will terminate, these individuals can also be very short.

Given the grammar presented in figure 4.3, if we choose the first rule $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ and we have an individual initialized to $[200, 48, 8, 36, 28, 60, 8, 104]$ we will loop infinitely on $\langle \text{expr} \rangle$ that is why we use a maximum wrapping limit. However is we chose $\langle \text{expr} \rangle ::= \langle \text{var} \rangle$ we are sure that this production will always lead to termination

```

(1) <expr> ::= <expr> <op> <expr> (A)
          | ( <expr> <op> <expr> ) (B)
          | <pre-op> ( <expr> ) (C)
          | <var> (D)

(2) <op> ::= + (A)
        | - (B)
        | / (C)
        | * (D)

(3) <pre-op> ::= Sin (A)
            | Cos (B)
            | Tan (C)

(4) <var> ::= X (A)

```

FIGURE 4.3 – Context Free Grammar Example

4.4 Mutation and Crossover

As already mentioned in our system we do not manipulate the phenotype, we rather manipulate the genotype which is a variable length linear string, due to this reason we use the canonical GA operators.

The mutation operator changes an integer by another one that is randomly generated, this is very important because when we use the grammar based GP [19] we cannot do it like this, we should respect the grammar and when mutating a certain node of an individual we go back to its father (upper node) then we generate randomly according to the upper node rule a new sub-tree. Applying the mutation at the genotypic level is more easy and interesting than applying it at the phenotypic level.

Crossover used in our system is the one point crossover, it swaps sections of the genetic code between parents [72]. However, because of the mapping process, the effect on the phenotype can be complex. The crossover used in this system is conducted under the same mechanics as in the GA. The genetic material to the right of the selected crossover that is applied at the genotypic level has a ripple effect on the derivation sequence after the crossover point and this is due to the mapping process [72], this is illustrated

in figure 4.4.

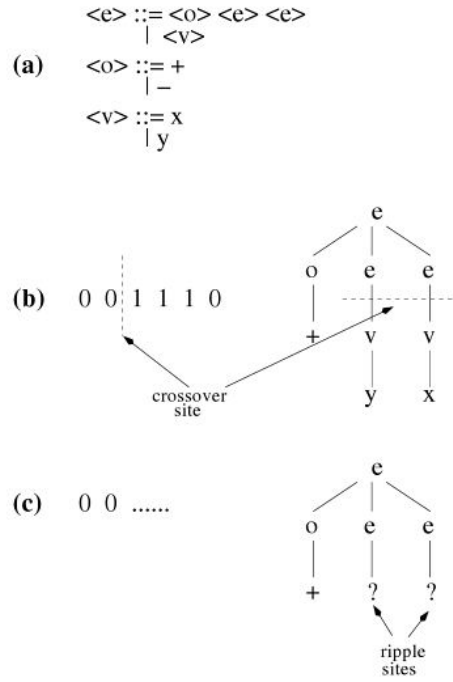


FIGURE 4.4 – Crossover Ripple effect

The grammar (a) in figure 4.4 is used to generate the tree (b), the crossover is applied on the genotype (binary string (b)). the genotypic sequence which is added after the crossover point will be placed in a context at the ripple sites which may differ from its original context.

4.5 Fitness Function and Individual Evaluation

Fitness function expresses how an individual i satisfies a certain problem. When using evolutionary algorithm to solve a certain optimization problem we should specify a function that will serve in measuring how good individuals of a certain population are. Individuals need to be evaluated so that the system will be able to choose the fittest ones to be reproduced, mutated and crossed over to generate new off springs.

In our system we manipulate arrays of integers and to be able to evaluate these arrays we need first to map them into their phenotypes, after mapping our individuals we will have complete rule based classifiers.

Figure 4.5 illustrates the mapping process. When mapping each individual we start with the first integer of the array and with the starting symbol of the grammar, we compute s the number of choices of this starting symbol then we enumerate each choice of this starting symbol from $[0]$ to $[s - 1]$, then a division is applied between the current integer and s ($integer_i / s$) the remainder of this division indicates the rule to be applied, then we move to the next integer $i + 1$ and the rule we will use now is the one that was selected by the previous integer, the same steps will be applied until the individual is entirely mapped, we can arrive to the end of the integers array but the rules are not mapped until the end in this case we will reuse the same array and this is due to its circular nature. However, if we will use wrapping infinitely we can fall in a infinite mapping process and therefor we always set a maximum number of wrappings (see figure 5.1 and figure 4.9).

After the mapping process is finished we will have a full algorithm that should be run in order to compute its fitness value. In our case we are dealing with rule based classifiers so the best way to compute the goodness of these latters is by computing their accuracy. However classification accuracy is not the only metric that could be used to compare classifiers. We should not that when using supervised learning such as classification we do not have performance metrics that will reliably assign better performance to the probabilistic true mode given finite validation data, as pointed in [71]. In a population j we will have acc_1 as the accuracy of the individual 1, acc_2 as the accuracy of the individual 2,...etc. To compare between two individuals we compare their accuracies and the greater the accuracy is the better the individual is. In order to compute a certain accuracy we need to specify a domain or a problem by using a data set. Our

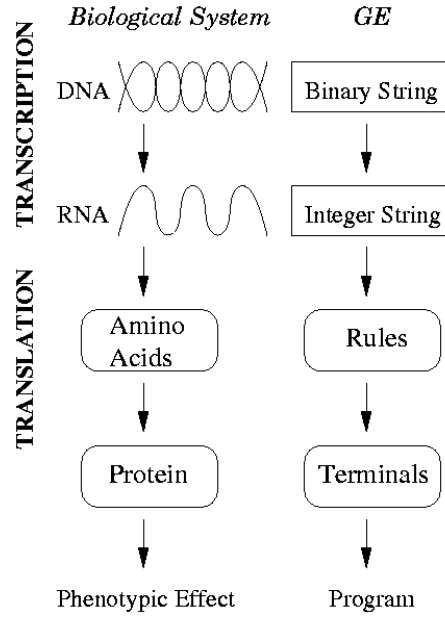


FIGURE 4.5 – The Mapping Process

system evolve general rule based classifiers therefor we should compute the accuracy of these latters over different data sets and that is why we have created a meta training set and a meta testing set. The meta training set consists of a set of data sets, each of them is divided into training and validation sets. To compute the accuracy of the individual we will compute its accuracy over each data set an then we will have a set of accuracies that will be averaged to give the overall accuracy of this individual as illustrated in figure 4.6.

4.6 The Grammar

1- $\langle \text{Start} \rangle ::= \langle \text{CreateRuleSet} \rangle \mid \langle \text{CreateRuleList} \rangle [\langle \text{PostProcess} \rangle]$.

$\langle \text{Start} \rangle$: generates one of the two rule models a rule set ($\langle \text{CreateRuleSet} \rangle$) or a rule list ($\langle \text{CreateRuleList} \rangle$). It also determines if the rule models generated will be post-processed or not.

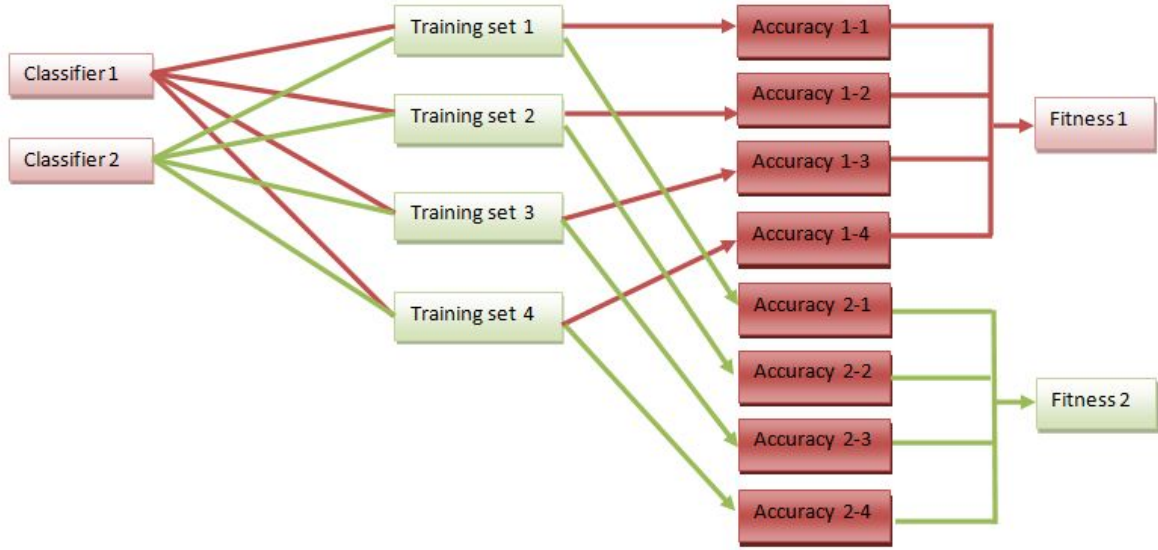


FIGURE 4.6 – The accuracy Computation

2- $\langle \text{CreateRuleSet} \rangle ::= \text{foreachClass } \langle \text{whileLoop} \rangle \text{ endFor } \langle \text{RuleSetTest} \rangle$.

3- $\langle \text{CreateRuleList} \rangle ::= \langle \text{whileLoop} \rangle \langle \text{RuleListTest} \rangle$.

$\langle \text{CreateRuleSet} \rangle$ and $\langle \text{CreateRuleList} \rangle$: describe the non-terminals CreateRuleSet and CreateRuleList. They represent the outer loops. The foreachClass terminal allows rules to be built for each class in turn.

4- $\langle \text{whileLoop} \rangle ::= \text{while } \langle \text{condWhile} \rangle \langle \text{CreateOneRule} \rangle \text{ endWhile}$.

$\langle \text{whileLoop} \rangle$: keeps adding rules to the rule set/list until condWhile is satisfied.

5- $\langle \text{condWhile} \rangle ::= \text{uncoveredNotEmpty} \mid \text{uncoveredGreater } (10 \mid 20 \mid 90\% \mid 95\% \mid 97\% \mid 99\%) \text{ trainEx}$.

$\langle \text{condWhile} \rangle$: is satisfied under one out of the two conditions : 1) When all the examples in the training set are covered by the current set of rules ; or 2) When a percentage of examples in the training set or a fixed number of them is covered by the current set of rules. We used 90%, 95%, 97% and 99% when working with percentages, and 10 or 20 examples when working with the absolute numbers. While the first condition requires that rules are produced for all the examples in the training set, the second

condition gives the algorithm some flexibility, and helps avoiding over-fitting.

6- `<RuleSetTest> : := lsContent | confidenceLaplace.`

7- `<RuleListTest> : := appendRule | prependRule.`

`<RuleSetTest>` and `<RuleListTest>` : define how the rules will be applied when classifying new instances. The rules in a decision list can be only applied in order. However, during the creation of the list, the rules can be appended or prep ended to it. The standard approach is to append rules to the list. In the case of rule sets, the rule defines which tie-breaking criterion will be applied in cases where two or more rules classify a test example (unseen during training) in two different classes. Among the options are the use of the ls-content of a rule or the Laplace estimation . Note that both these measures can also be used to evaluate rules when creating them, although ls-content is not used for this purpose in the current version of the grammar.

8- `<CreateOneRule> : := <InitializeRule> <innerWhile> [<PrePruneRule>] [<RuleStoppingCriterion>] .`

`<CreateOneRule>` : Rules are built following three basic steps : initialization, refinement and selection - the last two steps being iterative.

9- `<InitializeRule> : := emptyRule| randomExample| typicalExample| <MakeFirstRule>.`

`<InitializeRule>` : The first step generates an initial rule, a rule can be initialized in four different ways : (1) with an empty antecedent (represented by the terminal `emptyRule`), (2) from a seed example (picked randomly from the training set, and represented by the terminal `randomExample`) , (3) from a typical training example (represented by the terminal `typicalExample`) or (4) according to the frequency of the attribute-value pairs in the data set. An example is said to be typical if it is very similar to the other examples belonging to the same class it belongs to, and not similar to the other examples belonging to other classes. In other words, a typical example has high intra-class similarity

and low inter-class similarity. Eq. (4.1) shows how the typicality of an example is calculated. It is the ratio of the intra-class and interclass similarities, where the similarity between the examples e_1 and e_2 is calculated as the complement of the distance between the examples e_1 and e_2 , and P and N represent the number of positive and negative examples in the training set, respectively. The distance between the examples e_1 and e_2 is calculated as a simple Euclidean distance, as shown in Eq. (4.2). In Eq. (4.2), m represents the number of attributes in the data set, and max_i and min_i correspond to the maximum and minimum values assumed by the attribute i . In the case of nominal (or categorical) attributes, the distance with respect to a single attribute can only be 0 (if attribute values have the same value) or 1 (if attribute values have different values). If one of the attribute values is a missing value, the difference between it and any other attribute value is set to 0.5. In the case of initializing a rule according to the frequency of the attribute-value pairs, 10-

$\langle \text{MakeFirstRule} \rangle ::= \text{NumCond1} | \text{NumCond2} | \text{NumCond3} | \text{NumCond4}.$

$\langle \text{MakeFirstRule} \rangle$: creates a rule with 1, 2, 3 or 4 conditions in the rule antecedent. The pairs of attribute-values (conditions) inserted in the rule antecedent are selected using a probabilistic selection scheme, in which the probability of selecting a given attribute-value pair is proportional to the frequency of that attribute-value pair in the training set. After the initial rule is created using one of the methods described above, it is set as the current best rule, and then the rule refinement process starts. It is an iterative process that occurs inside the inner While loop .

11- $\langle \text{innerWhile} \rangle ::= \text{while} (\text{candNotEmpty} | \text{negNotCovered}) \langle \text{FindRule} \rangle \text{endWhile}.$

$\langle \text{innerWhile} \rangle$: follows the non-terminal InitializeRule in rule (8). As described in the conditions of the inner While loop, rules can be refined until they do not cover any negative examples, as stated by the terminal negNotCovered, or until a set of candidate rules is not empty (candNotEmpty). This set of candidate rules refers to the rules which

are undergoing the refinement process. At the first iteration, the only candidate rule is the one created in the initialization process. In the remaining iterations, they are the rules selected by the non-terminal `SelectCandidateRules` (rule 22), as will be explained later. At each iteration of the inner `While`, the non-terminal `FindRule`

12- `<FindRule> ::= <RefineRule> | <innerIf> <EvaluateRule> [<StoppingCriterion>] <SelectCandidateRules>`.

`<FindRule>` : creates a rule by : (a) finding all the possible refinements of the initial rule (`RefineRule`); (b) evaluating the rules generated through the refinements (`EvaluateRule`) and (c) selecting a fixed number of created rules to keep performing the refinement process. Moreover, `FindRule` also defines alternative ways to refine the rules (`innerIf`) and allows the definition of an alternative criterion to stop the rule refinement process (`Stopping Criterion`) .

17- `<RefineRule> ::= <AddCond> | <RemoveCond>`.

`<RefineRule>` : changes the current candidate rules by adding (`AddCond`) or removing (`RemoveCond`) conditions (attribute-value pairs) to/from them.

18- `<AddCond> ::= Add1 | Add2`.

19- `<RemoveCond> ::= Remove 1 | Remove2`.

`<AddCond>` and `<RemoveCond>` : According to the production rules generated by the `<AddCond>` and `<RemoveCond>`, either one or two conditions-at-a-time can be inserted in/removed from a rule. Most of the current rule induction algorithms, use a top-down or a bottom-up search while looking for rules. Only a few of them, like SWAP-1 [142], implement a bi-directional search strategy.

13- `<innerIf> ::= if <condIf> then <RefineRule> else <RefineRule>`.

14- `<condIf> ::= <condIfExamples> | <condIfRule>`.

15- `<condIfRule> ::= ruleSizeSmaller (2 | 3 | 5 | 7)`.

16- `<condIfExamples> ::= numCovExp ( >1 <)(90% | 95% | 99%)`.

<innerIf > : changes the way the rules are refined according to their size (<condIfRule >) or the number of examples covered by the current rule set/list (<condIfExamples >). In the former case, rules having size (number of conditions) smaller than 2,3, 5 or 7 might be refined in a different way than rules having size greater than or equal to 2, 3, 5 or 7. In the latter case, a decision is made based on the percentage of examples of the training set covered by the current rule set/list. It considers if the number of covered training examples is smaller or greater than 90%, 95% or 99% of the total number of examples in the training set. The motivation to include this inner!! statement in the grammar is that the algorithm can choose, for example, to add two conditions-at-a-time to the rule while its size is small (i.e. the number of examples covered by the rule is hopefully big enough to detect attribute interaction). However, as the rule size grows, and the number of examples covered by the rule shrinks, it might be easier to improve its predictive power just by adding one condition-at-a-time instead of two. The same argument is valid for the number of examples in the training set covered by the rule set/list. The fewer the examples left uncovered, intuitively the more difficult it is to find a combination of conditions which would improve the current rule.

20- <EvaluateRule> : := confidence | Laplace| infoContent| infoGain.

<EvaluateRule> :The rules created through the refinement process are evaluated using one of the measures defined in EvaluateRule, namely confidence, Laplace estimation, information content or information gain. These measures take into account the number of positive and negative examples the rule covers, and they will be used, in the last phase of the rule generation process, to select a subset of rules to go through further refinements. 21- <StoppingCriterion> : := MinAccuracy (0.6| 0.7| 0.8)| SignificanceTest (0.1|0.05| 0.025| 0.01).

<StoppingCriterion> :During the rule selection phase, the rule evaluation function can be combined with some other criterion in order to select the best rules found so far.

Hence, in some cases, a rule has also to fulfill a refinement stop criterion imposed by the optional non-terminal `StoppingCriterion`. `StoppingCriterion` requires a rule to have a minimum accuracy or to be significant according to a statistical significance test during the rule selection process. The minimum accuracy criterion calculates the accuracy of the rule and compares it with a threshold. The statistical significance test - a X2 (chi squared) test - calculates the distance between the distribution of the classes of the examples covered by the rule and the expected distribution (given by the frequencies of examples in each class for the entire training set). The lower the value of the distance, the higher the probability that the concept represented by the rule is due to chance [27].

22- `<SelectCandidateRules> ::= 1CR| 2CR| 3CR| 4CR| 5CR| 8CR| 10CR.`

`<SelectCandidateRules>` : The rule selection process, based on the values of the evaluation function and stopping criterion, selects a number of candidate rules to enter the next iteration of the rule refinement process. As specified in this rule, the number of selected rules can vary from 1 to 5, or it can be 8 or 10. Regardless of the number of selected rules, the current best rule is replaced only if its evaluation function value is worse than the value of the best selected rule.

23- `<PrePruneRule> ::= (1Cond| LastCond| FinalSeqCond) <EvaluateRule>.`

`<PrePruneRule>` : When this single rule building process terminates, a new rule is added to the rule set/list being produced. But before this happens, a last operation can be performed as a result of applying rule 8 : rule pre-pruning. `PrePruneRule` implements a pre-pruning method that tries to simplify a rule by removing one condition or a set of conditions from its antecedent. Rules can be simplified in three ways : (1) Removing one condition-at-a-time (`1Cond`) from its antecedent, as long as the new rule is better than the original rule according to an evaluation criterion ; (2) Removing the last added condition (`LastCond`) from its antecedent ; or (3) Removing a sequence of conditions from the end of the rule antecedent (`FinalSeqCond`), as long as the new rule is bet-

ter than the original rule according to an evaluation criterion. During the pre-pruning phase, rules are evaluated in a set of data different from the one used to build them. Recall that rules are created until all or a large part of the examples in the training set are covered by the generated rules. However, the process of rule creation can also be halted when a condition defined by the non-terminal RuleStoppingCriterion.

24- $\langle \text{RuleStoppingCriterion} \rangle ::= \text{accuracyStop } (0.5 \mid 0.6 \mid 0.7)$.

$\langle \text{RuleStoppingCriterion} \rangle$: is not satisfied. This condition is usually based on some property of the last rule found, and includes verifying if the accuracy (confidence) of the just-produced rule is greater than a threshold. We use threshold values of 0.5, 0.6 and 0.7.

25- $\langle \text{PostProcess} \rangle ::= \text{RemoveRule EvaluateModell } \langle \text{RemoveCondRule} \rangle$.

$\langle \text{PostProcess} \rangle$: Once the rule set/list is completed, the rule induction algorithm can still perform a last operation : post-process the rule model. The presence of a post-process step in the algorithm is determined by the application of NT 1. Post-processing methods (rule 25) can apply the same techniques used to pre-prune a single rule to all the rules in the rule model.

26- $\langle \text{RemoveCondRule} \rangle ::= (1\text{Cond} \mid 2\text{Cond} \mid \text{FinalSeq}) \langle \text{EvaluateRule} \rangle$.

$\langle \text{RemoveCondRule} \rangle$: similar methods to the ones described in rule 23. After the model is completed, it can be simplified by removing one (1Cond) or two (2Cond) conditions-at-a-time from the rule antecedents. In the case of rule sets, the model is simplified as long as the new rule is better than the original one according to an evaluation criterion. In the case of rule lists, this process goes on while the accuracy of the entire model is not reduced. The option of removing a final sequence of conditions is also available while post-processing rules. Besides, more compact models can be tried out by removing one by one entire rules from the current model, as implemented by the RemoveRule terminal in the right-hand side of the production rule generated by the rule

25. After each rule is removed from the rule set the whole model has to be re-evaluated, as indicated by the terminal EvaluateModel. Again, as in the pre-pruning phase, the rules/rule sets being post-processed are evaluated in a set of data different from the one used to build them. By applying the production rules defined by the grammar, we can generate up to approximately 5 billion different rule induction algorithms.

To summarize the what have been said about the system we present the following scheme(Figure 4.7) that describes the inputs and outputs of each module of the system, this can be helpful understanding the system in a more clear way. As we can see in (Figure 4.7) the first step in the process of automatically generating rule induction algorithms is the initialization that is performed by the initializer component which takes a file containing parameters to set the system ; parameters such as (mutation probability, maximum depth of trees, number of generations, crossover operation, grammar file, population size, fitness function, maximum number of wraps, selection operation, crossover probability, mutation operation, elite size and tournament size) then the initializer output a population of individuals (array of integers), these individuals are chromosomes of variable sizes.

The second step is the mapping process where the mapper takes as inputs the grammar file (Section 4.6) and the population of chromosomes generated in the previous step , the mapper also use some building blocks from the Weka (open source data mining tool written in Java) , these building blocks (basic operations of any classifier, like reading the examples and generating data statistics) were used in order to speed up the implementation process .The mapper proceeds by mapping of individuals (integer strings) into fragments of real machine codes (each individual will be converted into a rule induction algorithm) this is done by reading integers of the chromosome and using the modulo function as described earlier and deciding which non terminal or terminal of the grammar to choose , we should note that terminals of the grammar are associated

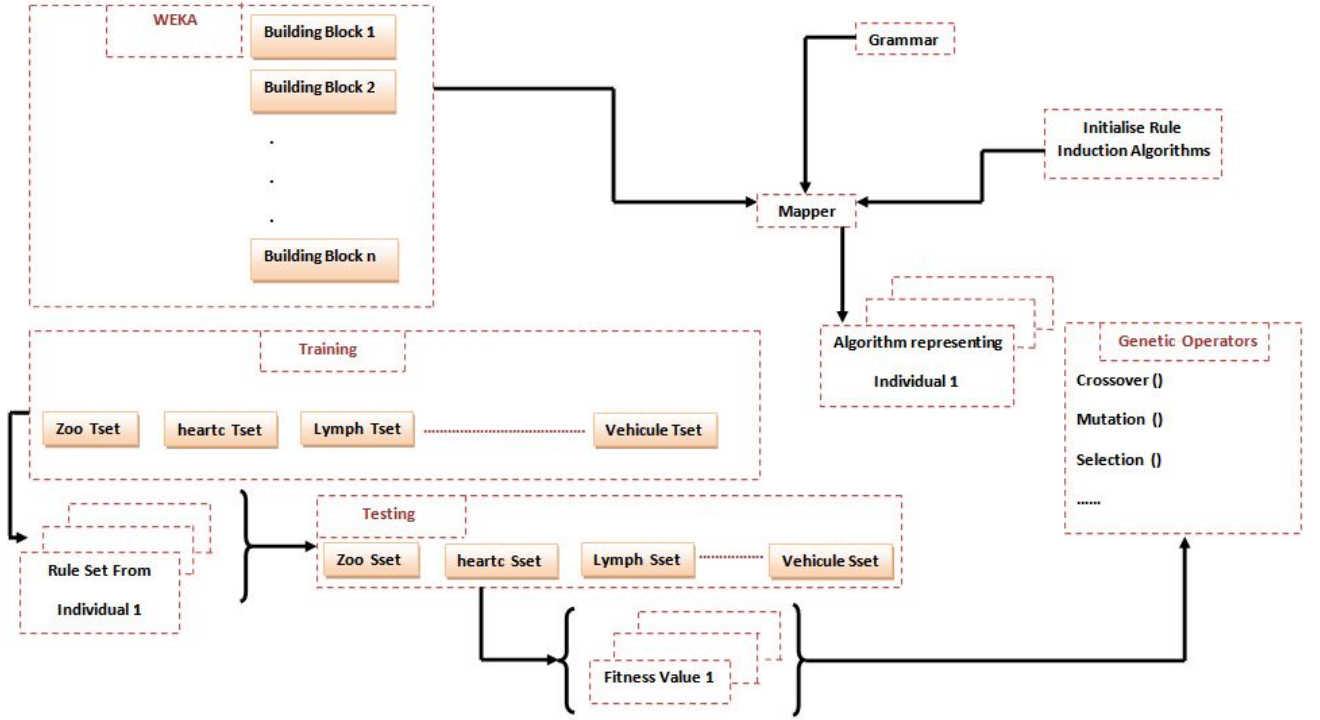


FIGURE 4.7 – Modules of The Proposed System

with Java codes(building blocks) after this the mapper import some basic operation of classifiers that are already coded in Weka then the resulting code can be executed in a set of classification problems and generate rule models for the corresponding data sets. The Third step is the evaluation of the resulting candidate algorithms the evaluator as we can notice in (4.7) takes only one input which is the population of individuals , these individuals will be run for each data set in the meta-training set, a set of classification rules will be generated from each training set and the accuracy of the algorithm is generated from each test set so for each individual (rule induction algorithm) we will have accuracy values to summarize these values into one fitness value we compute the average of the accuracies .For each individual in the population we will have a fitness value which will be used along with the parameters file with the genetic operators so that they'll be performed and a new population will be generated and the termination criterion is tested until it is satisfied and the fittest rule induction algorithm is found.


```

BEGIN:
E=dataset;
RuleList:=emptyset;
FOR each class C_{i} DO
  FOR each instance in E not covered DO
    Rule:= EmptyRule;
    PossibleRules:=emptyset;
    PossibleRules:=PossibleRules U Rule;
    WHILE rule still cover negative examples DO
      refinedrules:=emptyset;
      newPossiblerules:=emptyset;
      refinedrules:= Add1(rule);
      FOR each rule_{i} in refinedrules DO
        Laplace(rule_{i});
        IF rule_{i} is significant at level 0.05 DO
          newPossilerules:=newPossiblerules U rule_{i};
        END;
        IF Laplace(rule_{i}) is better than Laplace(Rule) Do
          Rule:= rule_{i};
        END;
      END;
      Possiblerules= 8 rules from newPossiblerules;
    END;
    RuleList=RuleList U Rule;
  END;
END;
END.

```

FIGURE 4.9 – Pseudo Code of the Individual Represented in Figure 5.1

	numClasses	Atts types	Domain
monks-1	2	nominal	computer science
monks-2	2	nominal	computer science
monks-3	2	nominal	computer science
mushrooms	2	nominal	life sciences
promoters	2	nominal	life sciences
splice	3	nominal	life sciences
vowel	11	nom/num	computer sciences
crx	2	nom/num	other
vehicle	4	numerical	imagery
pima	2	numerical	life sciences
heart-c	2	nom/num	life sciences
glass	7	numerical	physical sciences
zoo	7	nominal	life sciences
lymph	4	nominal	life sciences
balance scale desc	3	nominal	social sciences
ionosphere	2	numerical	physical sciences
segment	7	numerical	imagery
sonar	2	numerical	physical sciences
hepatites	2	nom/num	life sciences

TABLE 4.1 – Data Sets Description

The data sets were divided into 2 groups, 70% of them (13) were used to train then to validate the models, and the rest were used for the purpose of testing. We should note that this division was done randomly. The meta-training set contains (Monks-2, Monks-3, Balance-scale, lymph, promoters, splice, vowel, vehicle, pima, glass, sonar, hepatitis, ionosphere) and the meta-testing set contains (Monks-1, segment, crx, sonar, heart-c, mushrooms), before starting the training phase each set of the meta-training set is sub-sampled using the 5-fold cross-validation in order to avoid overfitting and to make predictions more generalizable[114]. The system needs 3 components to start the evolution process. The grammar mentioned earlier in section 3.2, the meta data sets and finally the grammatical evolution parameters, the number of generation was set to 40, the population size to 200, the mutation probability was set to 0.01, the crossover probability to 0.7, the selection method chosen is the tournament selection and we used the generational replacement. We should mention here that these parameters are not optimized but rather empirically chosen after analysing a certain number of trials. In order to evaluate the newly generated classifiers we computed the accuracies of 4 manually designed rule based classifiers using all 19 datasets. The first column of Table 2 reports accuracies of the new generated classifier (AGGE-classifier) while the remaining columns shows accuracy values of the 4 human designed classifiers (Ripper, Ridor, One-Rule and Prism), the first 13 rows reports the accuracies of the rules sets generated by the AGGE-classifier the 4 baseline. Using only the meta-training set (each row represent the test accuracy of a single set from the meta training set) these accuracy values are reported here to show the success of the training phase while the last six rows of the table shows the real predictive accuracy values because the AGGE-classifier has never met these sets during the training or validation phase.

The aim of this work was to propose a system that uses the grammatical evolution method to automatically generate rule based classifiers having an accuracy that can be at

	AGGE Classifier	Prism	Ripper	Ridor	OneRule
monks-2	53.8462	37.8698	53.2544	50.8876	42.0118
monks-3	36.8852	30.3279	45.9016	46.7213	37.7049
promoters	90.5660	66.0377	78.3019	74.5283	69.8113
splice	99.1223	70.1823	93.6991	92.1003	24.3574
vowel	82.2222	83.0303	69.6970	77.6768	31.8182
vehicle	64.4526	66.7849	68.5579	70.5674	51.4184
pima	75.3404	70.0349	75.1302	75.0000	70.1823
glass	62.6923	57.4939	66.8224	64.0187	58.4112
zoo	100.000	62.3762	86.1386	940594	42.5743
lymph	68.9189	75.6757	77.7072	85.1351	85.1351
balance scale desc	78.7225	52.3200	80.0800	79.5200	56.3200
ionosphere	88.1474	91.1681	89.7436	88.0342	80.9117
hepatites	68.3871	78.0645	78.0645	78.7097	83.2258
segment	94.2857	92.2944	95.4978	96.1472	64.8052
crx	91.4493	77.5362	85.5072	83.3333	85.7971
mushrooms	100.000	100.000	100.000	100.000	98.5229
monks-1	49.236	26.6129	49.1995	51.6129	50.0000
sonar	76.4423	74.0385	73.0769	73.5577	62.5000
heart-c	80.5291	76.8977	81.5182	79.538	71.6172

TABLE 4.2 – Accuracy rates (%) Using Both Meta-Sets

least competitive with the existing manually designed classifiers. After the implementation and the testing phase, the system proved its ability to produce classifiers that are highly competitive with the human designed ones. We can easily notice the performance of these formers in Table 1 , we should note that these accuracies were obtained by averaging the accuracy of the rule model generated by the AGGE-classifier for each test set over 5 iterations of the 5-fold-cross-validation method used during experiments, this also applies on the rest of the benchmark classifiers used for the purpose of comparison. It is worth mentioning that in Table 2 the new generated classifier has practically the same results as the other methods and if we compare only the baseline methods with each others we can clearly notice that the RIPPER records 10 wins over 5 for PRISM and RIDOR and 2 for the OneRule algorithm and this due to the sophisticated nature of the RIPPER classifier. It uses a growing, pruning and optimization phase and the Minimum Description Length (MDL) method as a stopping criterion when constructing the rules.

Now if we look at the AGGE-classifier accuracies we can notice how close are these accuracies to the baseline algorithms accuracies which is very interesting due to the fact that the AGGE-classifiers are automatically generated, and this removes a great deal of necessary time doing coding tasks. Human designers can easily go wrong when parameterizing an algorithm during the design process, contrariwise the chance of having bad parameters when using automatic evolution of algorithms is very low. The last six rows show that the AGGE-classifier records 3 wins against the baseline classifiers (crx, Mushrooms,sona), for the heart-c and segment data sets the AGGE results were very close the best accuracy (80.5291 versus 81.5182 and 94.2857 versus 95.4978) these results proved that the proposed approach can be very interesting. However, the AGGE system is time consuming while evolving AGGE-classifiers and requires high computational power. Moreover, it can't be run properly under an ordinary computer, where the

evolution process can take up to one week of continuous calculation, we should also note that this version of the system does not handle missing values and eliminates instances with missing value before using the datasets. Concerning numeric values the system uses the discretization method. To test the choices made by the AGGE system in matter of terminals, we tested the frequencies of these latters during each generation. Figure 4.10 illustrates the frequencies of the appearance of different rule refinement processes introduced by the grammar and these are : adding one attribute at a time (Add1), adding two attributes at a time to the rule (Add2), removing one attribute from the rule (Remove1) and removing two attribute at a time from the rule (Remove2), we can clearly see that at the first generation choices are random and the symbols dominating the population start to appear during the evolution process and as we can see (Add2) at the first generation (Initialization) was dominating the population then during the evolution it started to vanish, this is due to the random initialization in the first generation and to the elite choice during the new populations generation, we can also observe that the best refinements during the 20 generations are (Add1, Remove1), This maybe because removing or adding one attribute at a time is more stable and accuracy preserving than adding or removing two attributes at a time.

In figure 4.11 we can clearly see that the dominating rule evaluation measure is the (Laplace) one, the frequencies of this latter increases after each generation while the remaining measures frequencies decrease during the evolution.

Figure 4.12 presents the frequencies of terminals generated by the <InitializeRule> Non-Terminal, we can clearly notice that the way to initialize rules according to the AGGE System is by using the Empty rule as the initial one and this can be because when starting with an empty rule we will choose best attributes from scratch while using a typical example or a random one can be regarded a a refinement process and it may work or not.

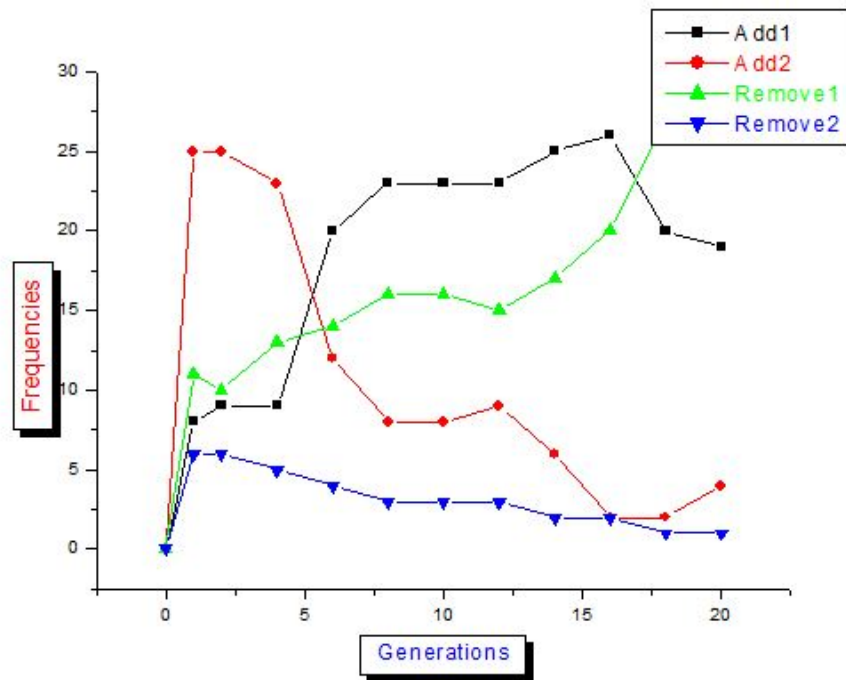


FIGURE 4.10 – Frequencies of Refinement Terminal Symbols During the 20 Generations

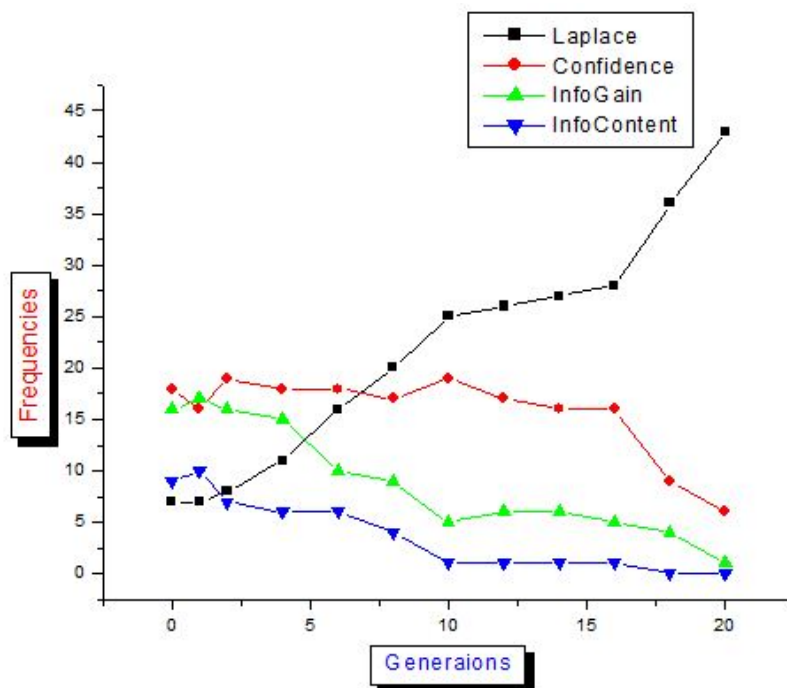


FIGURE 4.11 – Frequencies of Rules Evaluation Terminal Symbols During the 20 Generations

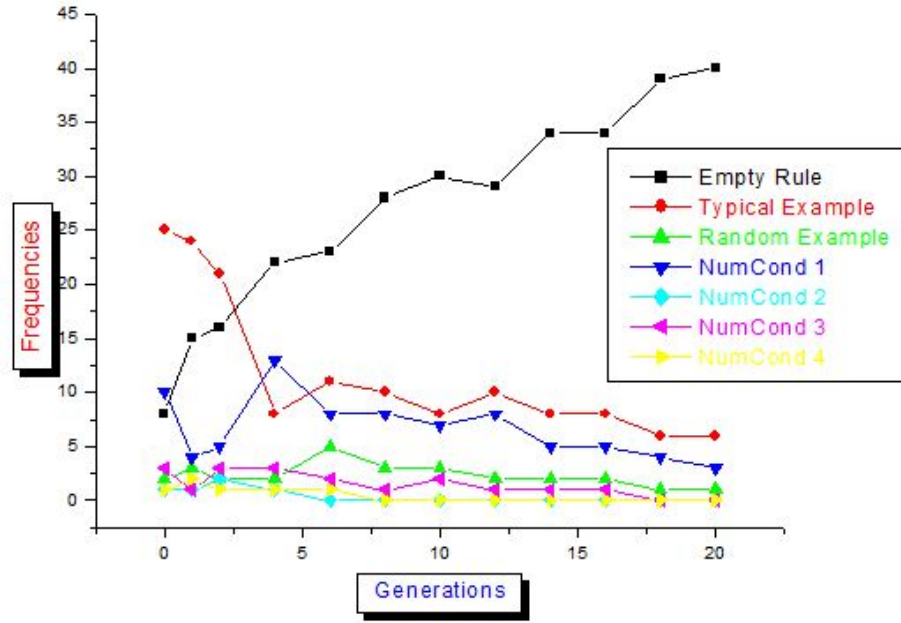


FIGURE 4.12 – Frequencies of Rules Initialization Terminal Symbols During the 20 Generations

We tested also different stopping criterion values of rule induction algorithms and the results are represented in figure 4.13, we can notice in he graph that the distribution is balanced between these different terminals, we can notice that from generation 10 (MinAccuracy 60%)and(MinAccuacy 70%) start to emerge and from generation 18 (MinAccuracy 70%) is the dominant one.

The main reason beyond choosing the grammatical evolution was to alleviate the burden of designing customized genetic operators faced when using the CFG-GP. Table 3 presents accuracies of classifiers evolved using our system (AGGE) in the first column and those evolved using (CFG-GP by [19]) in the second column. We should note that the parameters where set as follows : (generations=30, population size=100, crossover=0.7, mutation=0.25) for the CFG-GP system and (generations=20, population size=70, crossover=0.7, mutation=0.01) for the AGGE system. The upper 6 raws represent accuracy values using the meta training set and the lower ones using the meta

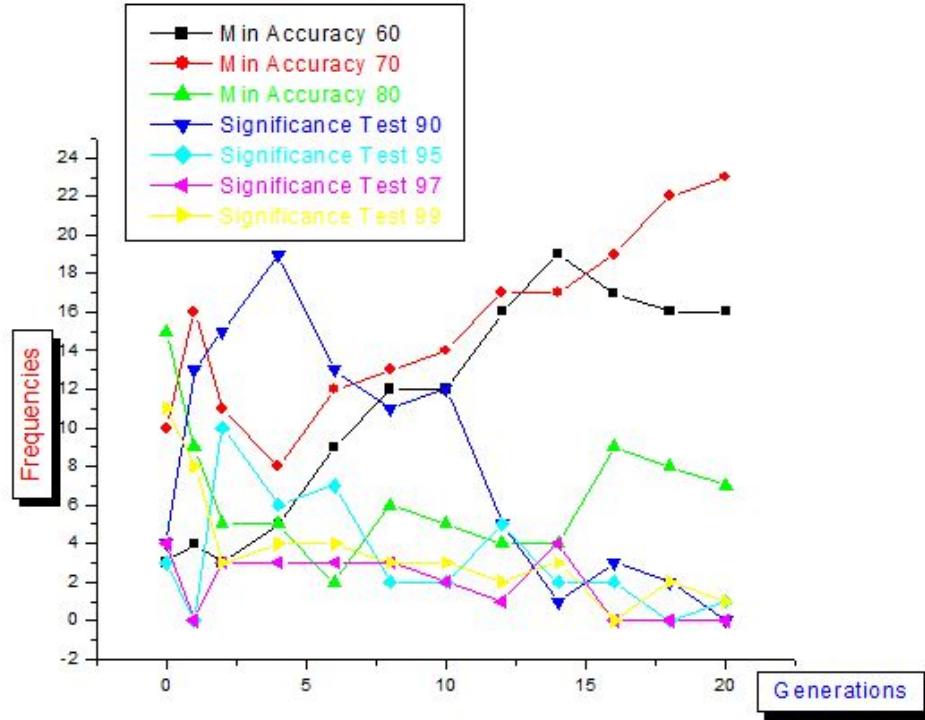


FIGURE 4.13 – Frequencies of the Algorithm stopping criterion Terminal Symbols During the 20 Generations

testing sets. We can notice that even though the size of the population and number of generations used in the AGGE system it classifiers recorded 8 wins versus 4 wins for the GGP classifier this is probably due to the ripple effect of the crossover operator of the AGGE system. the AGGE proved to be more interesting than the CFG-GP in terms of easiness of implementation and effectiveness of it's generated classifiers. However if we increase the number of generations or the mutation and crossover rates we might loose the efficiency of the system because of the destructive nature of it's operators so setting the AGGE parameters is highly sensitive.

	AGGE Classifier	GGP-Classifier
vowel	82.22	71.26
ionosphere	88.14	87.04
pima	75.34	70.18
zoo	100.00	96.70
hepatitis	68.38	70.28
glass	62.69	66.66
segment	94.28	95.06
crx	91.44	77.46
mushrooms	100.00	99.99
monks-1	49.23	99.93
sonar	76.44	72.34
heart-c	80.52	76.72

TABLE 4.3 – Accuracy Rates (%) of Automatically Evolved Classifiers Using Both Meta-Sets

4.8 AGGE Classifiers Versus Manually Designed Classifiers

The purpose of this work was to propose a system that is able to automatically evolve rule induction algorithms that are competitive with baseline ones. Results obtained after running and testing the system showed that the automatically evolved algorithms are highly competitive with manually designed ones.

In this section we will see the differences between the manually and automatically evolved ones. We will compare two algorithms that were automatically evolved by the AGGE system which performed well compared to baseline ones along with PRISM and RIPPER algorithms.

Algorithm (4.14) represents the PRISM pseudo code respecting the grammar’s lexis for the sake of comparison, the PRISM algorithm generates an unordered set of rules, it tries to produce rules that do not cover negative instance as quickly as possible, these

```

BEGIN:
  RuleList:=emptyset;
  FOR each class  $C_{\{i\}}$  DO

  REPEAT
    bestRule:= EmptyRule;
    CR:=emptyset;
    WHILE bestRule still cover negative examples DO
      CR:= add1condition to CR;
      Evaluate CR using Confidence;
      R= rule with maximum Confidence value;
    END
    RuleSet= RuleSet U R;
    Remove instances covered by R;
  Until all Examples from class  $C_{\{i\}}$  are covered;
  END;
END.

```

FIGURE 4.14 – Pseudo Code of the PRISM Algorithm

rules may have very small coverage. The PRISM algorithm starts with an empty rule and for each class it tries to build a set of decision rules that do not cover any negative example, we can see that the PRISM do not have a beam search as in the CN2 algorithm and it builds a rule by refining it using an operation that adds on condition (test) at a time, it also uses the confidence metric to evaluate the goodness of the generated rules. Algorithm 4.15) represents the RIPPER algorithm, it is clearly seen that it is a more sophisticated one, it builds a rule set by repeatedly adding rules to an empty rule set until all positive examples are covered. Rules are formed by greedily adding conditions to the antecedent of a rule (starting with an empty rule as the PRISM algorithm) Until no negative example is covered. After a rule set is covered an optimization stage is lead by post passing the rule set so as to reduce its size and improve its fitness to the training data. The minimum description length is used to stop rules adding to the rule set. During the optimization phase rules can either be revised or replaced. The next two algorithms

```

Let n be the number of classes
Sort classes in ascendent order, C1, ... ,Cn , according to their number of
examples
RuleSet = EmptySet;
FOR i = 1 to n-1 DO
    Positive = examples from Ci
    Negative = examples from remaining classes
    RuleSet2 = EmptySet
    REPEAT
        Divide the training data in Grow and Prune
        R = an empty rule
        WHILE R covers negative examples DO
            newCandidateRules = EmptySet;
            newCandidateRules = Add1condition to R;
            Evaluate newCandidateRules using information gain in Grow;
            R = best rule in newCandidateRules;
        END;
        R2 = Rule produced when removing last condition from R;
        WHILE (p-n)/(p+n)(R2) > (p-n)/(p+n)(R) in Prune DO
            R=R2;
            R2= Rule produced when removing last condition from R;
        END;
        RuleSet2 = RuleSet2 U R
        Remove examples covered by R from training set
    UNTIL Positive = 0 OR MDL of RuleSet2 is d bits > the smallest MDL
        found so far
    RuleSet = RuleSet U Optimized RuleSet2;
    Remove all examples covered by RuleSet2 from the training set;
END;
Make class C_{n} the default class

```

FIGURE 4.15 – Pseudo Code of the RIPPER Algorithm

(4.16 ,4.17) are rule induction algorithms that were automatically evolved using the AGGE system which was trained using different data sets from different domains. As we can clearly see Algorithm 4.16 creates an empty rule as in the PRISM and RIPPER seen previously, then it refines the rule according to it size if more than 5 it adds one condition at a time to the current rule else it adds two conditions at a time then it

evaluates the resulting refinements using the confidence metric and then it chooses the best one, if this latter have an accuracy greater than 0.8. Rules are created until no record is left in the data set. As we can see the difference between this algorithm and baseline ones is the refinement process where the new algorithm uses an if condition to choose between two refinement ways. When the rule set starts to be built, there is a lot of examples available in the data set, if the size of the current rule is less than 5 the algorithm tries to find the best combination of conditions (best in terms of confidence) else if the size is greater it adds the best condition to the rule and as a prepruning process the algorithms ignores rules with an accuracy less than 0.8. Algorithm 4.17 shows the pseudo code of a second algorithm generated by the AGGE system, this algorithm first starts the rule induction process by creating an empty rule and then according to the data covered it decides whether to add one or two conditions at a time after that the new refined rules is evaluated using the confidence metric to choose the best refinement after that a rule is refined, it tries to preprune it by removing the last condition then evaluate the new pruned rule and test it against the original one and chooses the best one as the best rule to add it to the rule list. This process will be repeated until all data is covered.

4.9 State of The Art

If we go through the literature we find different attempts to automate different the generation of some parts of certain algorithm, in 1996 Cho [75] proposed a system to evolve artificial neural networks, his system have the ability to adapt to an environment as well as the changes in this environment.

In 1998 Suyama [76] proposed a system called CAMLET. This system combines GP with local search methods to evolve a classification algorithm, this system uses an ontology

```

BEGIN:
  RuleList:=emptyset;
  FOR each class C_{i} DO

  REPEAT
    bestRule:= EmptyRule;
    CR:=emptyset;
    CR:=CR U bestRule;
    REPEAT
      For each rule i in CR DO
        newrules= emptyset;
        IF rule size > 5
          add1condition to rule;
        Else
          add2conditions to rule;
        Evaluate rule using Confidence;
        IF accuracy(rule)> 0.8
          newrules= newrules U rule;
        ENDfor
      CR= 3 best rules from newrules;
    Until CR = EmptySet;
    RuleList=RuleList U bestRule;
    Remove Examples Covered by bestRule;
  Until all Examples are covered;
  ENDfor;
END.

```

FIGURE 4.16 – Rule Induction Algorithm Generate Using AGGE system

to guide the search and not a grammar. The ontology used in CAMLET have 15 coarse gained building blocks and its leaf nodes are full classification algorithms. Suyama used the accuracy of individuals as the fitness function of this system.

In 1998 Wong [74] also attempted to automatically evolve the evaluation function of the FOIL (first order inductive logic programming) algorithm using the GGP system. The system proposed in [74] tries to evolve a population of scoring functions (evaluation functions) following production rules of a logical grammar. In this system, initialization can be done by the user to introduce prior knowledge to it, it can also be done using

```

BEGIN:
  RuleList:=emptyset;

  REPEAT
    bestRule:= EmptyRule;
    CR:=emptyset;
    CR:=CR U bestRule;
    REPEAT
      For each rule i in CR DO
        newrules= emptyset;
        IF 95% of dataa is covered
          add1condition to rule;
        Else
          add2conditions to rule;
        Evaluate rule using Confidence;
        newrules= newrules U rule;
      ENDfor
      CR= 3 best rules from newrules;
    Until CR = EmptySet;
    PrunedRule= remove last condition from bestRule;
    Evaluate pruned rule using Confidence;
    If PrunedRule better than bestRule
      bestRule= PrunedRule;
    RuleList=RuleList U bestRule;
    Remove Examples Covered by bestRule;
  Until all Examples are covered;

END.

```

FIGURE 4.17 – Rule Induction Algorithm Generate Using AGGE system

another learning system the individuals of each population are used by a generic version of a top down first order logic learning algorithm and the whole learning algorithm is evaluated by computing the number of miss-classified examples. Wong used 4 learning taks to evaluate its system.

in 1999 , Evolutionary Algorithms have been used by Yao [73] to automatically evolve neural networks, first EA's where used to evolve only the training weights, the number of hidden layers, number of hidden neurons and the interconnections between neurons[85].

In 2003 Yamaguchi [77] proposed a new version of CAMLET, this new version is a parallelized one and it does not use the hybrid search method proposed by Suyama, it uses a simple genetic algorithm. Yamaguchi used mutation, crossover and the tournament selection as genetic operators.

In 2007 Gisele [19] used an individual encoding solution GGP system to automatically evolve rule induction algorithms. The GGP proposed by Gisele uses a grammar describing the overall structure of rule induction algorithms following sequential covering strategy, individuals are represented using a tree like representation and they are initialized using the production rules of the grammar. different fitness functions were used using a single objective and a multi objective fitness function to evaluate algorithm generated by her system, Gisele used 20 data set from different domains and applications.

4.10 Summary

This chapter introduced a system that is grammatical evolution based the system was used to automatically evolve rule induction algorithms. A detailed description of the way this system works is presented in this chapter along with its main components and how they are used. Individual representation, population initialization, Fitness function and genetic operators are also reported here.

We also presented the grammar describing the sequential covering paradigm and we explained each of its rules, a quick state of the art in rule induction area is introduced along with the system description in order to localize our work. Different experiments that were conducted using the AGGE system are reported also in this chapter with a discussion of these results. In the next chapter we will introduce an extension of this system, this new system extension is used to automatically induce fuzzy rule based classifiers.

CHAPITRE 5

Automatic Induction of Fuzzy Rule Based Classifiers FuAGGE

5.1 Introduction

In the previous chapter we developed a system that can automatically evolve sequential covering algorithms. These algorithms proved to be very efficient, robust and competitive with baseline ones. In this chapter we address the fuzzy set covering method which is an extension of the sequential covering one. We will review the extension of the previous AGGE system, this extension will be noted as FuAGGE and it stands for the Fuzzy Automatic Generator of rule based classifiers using Grammatical Evolution. The aim of this system is to evolve computer programs that are able to build concepts descriptions by inducing conjunctive expressions matching a subset of positive Instances. We choose fuzzy rule based classifiers because fuzziness increases the expressiveness, human comprehensibility and representation power by supporting vagueness, ambiguity and uncertainty. In this chapter we will be dealing with the fuzzy set covering method which is a separate-and-conquer and general-to-specific method and it can be considered as the fuzzy version of the sequential covering method. In 1965 Lotfi Zadeh [29] proposed fuzzy sets as a generalization of crisp sets to address the incapability of crisp sets to model uncertainty and vagueness inherent in the real world. Initially, Fuzzy sets did not receive a very warm welcome as many academics stood skeptical toward the theory of "imprecise" mathematics. In the middle to late 1980's the success of fuzzy controllers

brought fuzzy sets in the limelight and many application started appearing [41] such as Fuzzy trees [42,83], Fuzzy clustering [43,84], Fuzzy NNets [44] and for each sequential covering classifier a fuzzy extension was proposed e.g. Fuzzy AQR in 2003 [45], in 2004 appeared the FuzzConRi [46] a fuzzy classifier representing the fuzzy CN2, In 2005 the FuzzyBexa was proposed [47], in 2009 FURIA the fuzzy extension of the famous RIPPER algorithm appeared [48] and the FILSMR appeared in 2010 and it represent the fuzzy PRISM classifier [49].

5.2 Fuzzy Set Theory

Let U be a given universal set. Generally, a set A , $A \in U$, is defined using one of three methods : listing each element in the set, e.g. $A = a, b, c$, using a proposition to describe a property that must be satisfied by all the members of the set, e.g. $A = x | x \in Z, 0 < x < 10$, or using a function, usually called the characteristic function, that declares which elements are members of the set, $\mu_A(u) = 1$, for $u \in A$, $\mu_A(u) = 0$, for u not belonging to A where $u \in U$. Fuzzy sets are a generalization of crisp sets, and are defined using the functional method, where the characteristic function is defined as $\mu_A(u) : U \longrightarrow [0, 1]$ The degree to which an element u , $u \in U$, belongs to the fuzzy set A is described in terms of the membership function $\mu_A(u)$. This degree of membership expresses the certainty or ambiguity that u belongs to A , with $\mu_A(u) = 1$ meaning absolute certainty that $u \in A$, and $\mu_A(u) = 0$ absolute certainty that u does not belong to A . Crisp sets are special cases of fuzzy sets, since for a crisp set, $\mu_A(u) : U \longrightarrow 0, 1$ i.e. the membership function is either 1 or 0, and elements can either belong to a set or not with absolute certainty.

Attributes are usually of two types : Nominal : which takes a finite set of unordered values (e.g. attribute outlook takes the values sunny, cloudy, and rainy).

Real : which take values from a linearly ordered range (e.g. temperature).

For real attributes the fuzzy membership function maps the linear domain to membership degrees on the scale $[0, 1]$. The figure below shows how temperature values are mapped onto membership degrees for the term set of temp, defining the membership functions μ_{cold} , μ_{mild} and μ_{hot} [79,82].

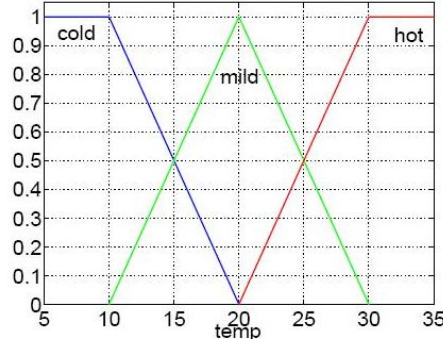


FIGURE 5.1 – An Example of Memberssship Functions

Linguistic variables with an unordered input domain, for example outlook (sunny, cloudy, rainy) have no associated mapping from a linear domain to membership degrees. In this case the membership function just describes the ambiguity that an instance belongs to a certain term.

5.3 Fuzzy Rule Set Learners

Since the fuzzy set theory appeared the researchers are trying to find fuzzy versions and fuzzy alternative algorithms for all data mining problems and sub-fields we state as examples the fuzzy inductive learners, the fuzzy decision trees[104,107] and the fuzzy clustering method. In this work we are interested in the sequential covering rule induction paradigm. As already mentioned the fuzzy version of the sequential covering paradigm is called fuzzy set covering. In this section we will review some of the fuzzy

set covering algorithms which are of course the fuzzy version of sequential covering rule set inducers and then we will introduce a grammar that presents the overall structure of these latters. This grammar will be eventually used by the system to automatically generate fuzzy rule learners.

5.3.1 Fuzzy-AQR

AQR is an inductive learning system that uses the basic AQ algorithm to generate a set of classification rules. When building classification rules, AQR performs a heuristic search through hypothesis space to determine the descriptions that account for all positive instances and no negative instances. AQR processes the training instances in stages; each stage generating a single rule, and then removing the instances it covers from the training set. This step is repeated until enough rules have been found to cover all instances in the chosen class. Since data in real-world applications usually contain linguistic information, conventional inductive learning procedures may be inapplicable to some real domains. Fuzzy concepts can then be applied to such conventional inductive learning approaches[86]. The fuzzy inductive learning task is thus to find a concept description R such that the following conditions are met, [45] :

$$\forall e \in_{\beta} P \Rightarrow e \subset_{\alpha} R, \text{ and } \forall e \in_{\beta} N \Rightarrow e \not\subset_{\alpha} R$$

Figure 5.2 represents the pseudo code of the Fuzzy-AQR algorithm.

The fuzzy AQR learning strategy consists of two main phases : generation and testing. The generation phase generates and collects possible fuzzy complexes into a large set ; the testing phase then evaluates each element in this set according to the value of $U(a+) + (a-)$. The best fuzzy complex as an extra disjunct is then added to the set of concept descriptions. This procedure is repeated until all soft positive instances in P_{β}

```

BEGIN:
FruleSet:=emptyset;
ForEach Concept Ci
  pos= {Instances in data | Uci(instance)> alpha-cover};
  neg=data-P;
  WHILE not (pos = emptyset) Do

    Select SEED (highest membership Uci(instance));
    Ant={single selector complexes}
    AntSet= Selectors Si in Ant | Si alpha-cover(SEED) ;

    WHILE( Si exists in AntSet)&&(Si alpha- cover(Neg)) DO

      Compute Uexclude(Si) in AntSet;
      AntEx= Select Min(Uexclude(Si,AntSet));
      Inst= I in Neg,Uneg(I), I alpha-covered(AntEx);
      Set={Si |Si alpha-cover(SEED) && Si not alphacover(Inst) };
      AntSet={AntSeti and Setj |AntSeti is the oldAntSeti ,Setj in Set};
      AntSet= AntSet - Ci that subsume others;
      AntSet= best 5 (AntSet);
      Compute Uinclude(AntSeti);
      Compute Uexclude(AntSeti);
      Compute U(a+)+(a-);

    DONE;
    Select bestAnt= AntSet with Max(U(a+)+(a-));
    FruleSet= FruleSet+ best AntSet;
    Remove from Pos, Neg the instances covered;

  DONE;
ENDfor;

END.

```

FIGURE 5.2 – Fuzzy-AQR Pseudo Code

have been α – covered by the set of concept descriptions[45].

5.3.2 Fuzzy-CN2 (FuzzConRI)

FuzzConRI is a fuzzy rule induction algorithm based on the CN2 [80] rule set inducer. It consists of two layers, an upper layer implementing a set cover approach to rule induction, and a lower layer for inducing one rule. The upper layer receives a set of training instances T , and a set of concepts $Concepts$. It starts by initializing the rule set to the empty set, and then considers each concept $2\ Concepts$, one at a time. The training set T is split into two parts, a set of fuzzy instances P that belong to the concept and a set of fuzzy instances N that do not belong to the concept. The sets P and N are called the set of positive and negative instances, respectively [46]. The procedure `FindBestAntecedent` is then repeatedly called to induce a conjunction that covers a set of instances in T . The instances in P covered by the conjunction are removed from P and the procedure repeated until P is empty[45]. Figure 5.3 represents the pseudo code of the Fuzzy-CN2[81] algorithm and Figure 5.4 represents the pseudo code of the `FindBestAntecedent` procedure.

5.3.3 Fuzzy PRISM (FILSMR)

FILSMR is a fuzzy inductive learning algorithm based on the PRISM learning strategy that handles vagueness and eliminates irrelevant tests occurring in the rule. This learning algorithm maximizes fuzzy information gain instead of minimizing entropy in inducing modular rules. It concentrates on finding relevant attribute-value pairs, instead of just attributes. During induction, the actual amount of information contributed by each attribute-value pair (selector) is evaluated for a specific classification, and the one with the maximum fuzzy information gain is then selected and added into the induced rule.

```

BEGIN:
  FruleSet= emptyset;
  FOReach concept Ci

    p=instances alpha belonging to Ci;
    n=data-p;
    WHILE (not(p= 0)) DO

      Ant=FindBestAntecedent(p,n);
      IF ( suitable Ant exists)

        create rule {if bestAnt then Ci};
        FruleSet=FruleSet U rule;
        data=data-alpha-covered(rule,data);

      ENDif;
    DONE;
  ENDfor;

END.

```

FIGURE 5.3 – Fuzzy-CN2 Pseudo Code

```

BEGIN:
  Ant={True};
  bestAnt= emptyset;
  Terms={attribut values existing in data};
  WHILE (not(Ant=emptyset)) DO
    AnSet={<x,y> | x in Ant and y in Terms};
    AntSet=AntSet-Ant;
    Laplace(AntSet);
    Foreach Si in AntSet( suitable Ant exists)
      IF (Si betterThan bestAnt) // highest Laplace value
        bestAnt=Si;
      ENDif;
    ENDfor;
    AntSet=AntSet- Si covering only positives
    Laplace(AntSet);
    AntSet=selectBest(beamsize,AntSet);
    Ant=AntSet;
  DONE;

END.

```

FIGURE 5.4 – FindBestAntecedent Procedure Pseudo Code

Each selector can be thought of as a message, and the classification a δ_k can be also thought of as an event. Given a message S_i , the amount of fuzzy information gain about an event δ_k is defined as :

$$I(\delta_k|S_i) = \log_2(H(\delta_k|S_i)) - \log_2(H(\delta_k))$$

where $H(\delta_k|S_i)$ and $H(\delta_k)$ are respectively the subsequent and antecedent fuzzy information and are defined in [49]. Figure 5.5 represents the pseudo code of the Fuzzy-PRISM algorithm.

```

BEGIN:
  FruleSet= emptyset;
  FOReach concept Ci

    p=instances belonging to Ci;
    n=data-p;
    WHILE (not(p= 0))&&( alpha-belonging)  DO

      newAnt= empty;
      newAnt= addSelectors;
      1: FORall newAnt
        compute InfoGain;
      END;
      Select bestAnt;
      Compute B(Ci,bestAnt);
      IF (B > beta-belonging)

        create rule {if bestAnt then Ci};
        data=data-alpha-covered(rule,data);

      ELSE
        data2= instances alpha-cover(bestAnt);
        go to 1;
      ENDif;
    DONE;
  ENDfor;

END.

```

FIGURE 5.5 – Fuzzy-PRISM Pseudo Code

5.4 The System Settings

The approach used in the FuAGGE system to represent individuals is the same as the one used in the AGGE system as it is the same system. Individuals are represented as arrays of integers and then are mapped to possible solutions (possible fuzzy rule learners). These latters are called genotypes. The system initializes individuals as strings of random integers, when mapping these individuals we have no guarantee that this will terminate neither we have a guarantee that the strings are long enough. In the first case we do not have any problem the mapping process will stop reading the integer when it arrives to a rule that do not have non terminals. As for the second case we have the wrapping method where the mapper loops over the individual as many times as it is necessary to terminate a full mapping process. As we are manipulating strings of integers we do not have to worry about the genetic operators, the operators are used in the simplest way possible where the mutation operator changes randomly an integer by another one. While the crossover swaps sections of two parents (a one point crossover is used). The genetic operators are conducted under the same mechanism as in the standard GA's. Fitness function calculates the goodness of possible solutions by testing their accuracy. possible solution represent fuzzy rule learners. To test the efficiency of a rule learner we should test its classification accuracy over a number of datasets. In a population j we will have acc_1 as he accuracy of the individual 1, acc_2 as the accuracy of the individual 2,...etc. To compare between two individuals we compare their accuracies and the greater the accuracy is the better the individual. In order to compute a certain accuracy we need to specify a domain or a problem by using a data set. Our system evolve general fuzzy rule based classifiers therefor we should compute the accuracy of these latters over different data sets and that is why we have created a meta training set and a meta testing set. The meta training set consists of a set of data sets, each of them is divided into training and validation sets. To compute he accuracy of the individual

we will compute its accuracy over each data set and then we will have a set of accuracies that will be averaged to give the overall accuracy of this individual

5.4.1 Fuzzy Set Covering Grammar

After presenting components and pseudo codes of some of the existing rule set inducers following the fuzzy set covering paradigm, we will introduce the grammar that represents the components and the general structure of such algorithms, this grammar was formed after reviewing different fuzzy rule set inducers in the existing literature. It will be then used with our system in order to generate rule set inducers that are able to produce fuzzy rule sets, after that we will present the results obtained by the system.

The Grammar contains 19 rule :

1- $\langle \text{Start} \rangle := \langle \text{FuzzyRuleSet} \rangle [\langle \text{PostPrune} \rangle]$

$\langle \text{Start} \rangle$: Generates a fuzzy rule set by the rule $\langle \text{FuzzyRuleSet} \rangle$ and it also determines if the rule models generated will be post processed or not.

2- $\langle \text{FuzzyRuleSet} \rangle := \text{For-each-concept } \langle \text{WhileLoop} \rangle \text{ endFor.}$

$\langle \text{FuzzyRuleSet} \rangle$: Describes the non terminal FuzzyRuleSet. It represents the outer loop of the fuzzy rule induction algorithms. The forEachClass terminal allow rules to be built for each class in turn.

3- $\langle \text{WhileLoop} \rangle := \text{While } \langle \text{Cond} \rangle \langle \text{GenerateAnt} \rangle \text{ endWhile}$

$\langle \text{WhileLoop} \rangle$: Keeps generating rules using the $\langle \text{GenerateAnt} \rangle$ rule and adding it to the fuzzy rule set until $\langle \text{Cond} \rangle$ is not satisfied.

4- $\langle \text{Cond} \rangle := (\text{Pos} \geq \langle \alpha - \text{cover} \rangle) \neq \emptyset \mid \alpha - \text{Uncover} \geq (90\% \mid 95\% \mid 97\% \mid 99\%)$

$|((\text{Pos} \geq \langle \alpha - \text{cover} \rangle) \neq \emptyset \text{ (newAntExist)})$

$\langle \text{Cond} \rangle$: is satisfied under one out of three condition : (1)- When all examples in the training set are $\alpha - \text{covered}$ by the current set of the fuzzy rules. or (2)- When a percentage of examples in the training set is $\alpha - \text{covered}$ by the current set of fuzzy rules. We used 90, 95, 97, 99 percentages. (3)- When all examples in the training set are $\alpha - \text{covered}$ by the current set of the fuzzy rules and there exist another antecedent to be added. While the first and the third conditions requires that fuzzy rules are produced for all the examples in the training set, the second condition gives the algorithm some flexibility, and helps avoiding over-fitting.

5- $\langle \text{GenerateAnt} \rangle$: $:= \langle \text{InitializeAnt} \rangle \langle \text{SecWhile} \rangle [\langle \text{PrePrune} \rangle]$

$\langle \text{GenerateAnt} \rangle$: Rules are built following three basic steps : initialisation, refinement and selection. the last two steps being iterative.

6- $\langle \text{InitializeAnt} \rangle$: $:= \text{True} \mid \text{randomSEED} \mid \text{bestSEED}$

$\langle \text{InitializeAnt} \rangle$: The first step generates an initial fuzzy rule, a rule can be initialized in three different ways : (1) with the most general antecedent (represented by the terminal True), (2) from a seed example (picked randomly from the training set, and represented by the terminal randomExample), (3) from a typical training example (represented by the terminal bestSEED).

7- $\langle \text{SecWhile} \rangle$: $:= \text{While } \langle \text{SecCond} \rangle \langle \text{FindAnt} \rangle \text{ endWhile}$

$\langle \text{SecWhile} \rangle$: follows the non-terminal GenerateANT in rule (5). A set of Fuzzy candidate rules that refers to fuzzy rules which are undergoing the refinement process. At the first iteration, the only candidate rule is the one created in the initialization process.

8- $\langle \text{SecCond} \rangle : := \text{SelectorSetNotEmpty} \mid \text{SelectorSetNotEmpty} \quad \text{SelectorNeg} \langle \alpha - cover \rangle$

$\langle \text{SecCond} \rangle$: As described by SecCond the conditions of the second while are ,(1)- fuzzy rules can be refined until a set of possible selectors is empty (SelectorSetNotEmpty) or (2)- fuzzy rules can be refined until the set of possible selectors is empty and no selector $\alpha - cover$ any negative examples, as stated by the terminal SelectorNeg $\langle \alpha - cover \rangle$.

9- $\langle \text{FindAnt} \rangle : := \langle \text{RefineAnt} \rangle \langle \text{EvaluateAnt} \rangle \langle \text{SelectAnt} \rangle \mid \langle \text{RefineAnt} \rangle \langle \text{Bayes} \rangle$
 $\mid \text{if numCovExp} (>1 <) (90\% \mid 95\% \mid 99\%) \text{ then } \langle \text{RefineAnt} \rangle \text{ else } \langle \text{RefineAnt} \rangle \mid \text{if ant-SizeSmaller} (2 \mid 3 \mid 5 \mid 7) \text{ then } \langle \text{RefineAnt} \rangle \text{ else } \langle \text{RefineAnt} \rangle$

$\langle \text{FindAnt} \rangle$: Creates a fuzzy rule by : (a) finding all the possible refinements of the initial fuzzy rule (RefineAnt); (b) evaluating the fuzzy rules generated through the refinements (EvaluateAnt) and (c) selecting a fixed number of created fuzzy rules to keep performing the refinement process. Moreover, FindAnt also defines alternative ways to select refined rules which is by using $\langle \text{Bayes} \rangle$ rules where the all possible refinements are evaluated according to the fuzzy Bayes measure[82]. The third and the forth rules changes the way the rules are refined according to their size (AntSizeSmaller) or the number of examples covered by the current fuzzy rule set (numCovExp). In the former case, rules having size (number of antecedents) smaller than 2,3, 5 or 7 might be refined in a different way than rules having size greater than or equal to 2, 3, 5 or 7. In the latter case, a decision is made based on the percentage of examples of the training set covered by the current fuzzy rule set. It considers if the number of covered training examples is smaller or greater than 90%, 95% or 99% of the total number of examples in the training set.

10- $\langle \text{RefineAnt} \rangle : := \langle \text{IncludeSelectors} \rangle \mid \langle \text{ExcludeSelectors} \rangle$

<RefineAnt> : Changes the current fuzzy candidate rule by adding (IncludeSelectors) or removing (ExcludeSelectors) conditions (attributs-value pairs) to/from them.

11-<IncludeSelectors> : :=Include1selector | Include 2selectors

12-<ExludeSelectors> : :=Exlude1selector | Exlude2selectors

<IncludeSelectors> and <ExludeSelectors> : According to the production rules generated by the IncludeSelectors and ExludeSelectors, either one or two conditions-at-a-time can be inserted in/removed from a rule. Most of the current rule induction algorithms, use a top-down or a bottom-up search while looking for rules.

13-<EvaluateAnt> : := LaplaceEstimate | FuzzyLaplace| FuzzyLs-content| FuzzyAcc-Function| FuzzyPurity| FuzzyInfGain| FuzzyEntropy

<EvaluateAnt> : The fuzzzy rules created through the refinement process are evaluated using one of the measures defined in EvaluateAnt, namely LaplaceEstimate (see 5.1), fuzzyLaplace (see 5.2), fuzzy Ls-Content (see 5.3), fuzzy Accuracy function (see 5.4), fuzzy purity (see 5.5), fuzzy information gain (see 5.6), fuzzy entropy (see 5.7). These measures take into account the number of positive and negative examples the rule α – covers, and they will be used, in the last phase of the rule generation process, to select a subset of rules to go through further refinements. Equations are cited below where p,n,P and N are integer numbers. p : the number of positive instances covered, n : the number of negative instances covered, P : the number of positive instances and N : the number of negative instances, where an instance is either covered with a membership μ_α or above, or not covered A : is a fuzzy antecedent, B : is a fuzzy consequent and c : is the conjunctin describing A.

$$M(S, c) = \sum_{i \in X_s(c)} mu_c(i)$$

where $X_s(c)$ means all instances in the set S that match c with membership μ_α .

$$L(r) = \frac{p+1}{p+n+nbClasses} \quad (5.1)$$

$$F(r) = \frac{\sum_{i \in X_T(c)} \mu_{A \cap B}(i) - \frac{1}{2}}{M(T, c)} \quad (5.2)$$

$$Lsc(r) = \frac{\frac{p+1}{p+2}}{\frac{n+1}{n+2}} \quad (5.3)$$

$$A(r) = M(P, c) - M(N, c) \quad (5.4)$$

$$P(r) = \frac{\sum_{u \in U} \min(\mu_A(u), \mu_B(u))}{\sum_{u \in U} \mu_A(u)} \quad (5.5)$$

$$Fic(r) = \log M(P, c) - \log M(N, c) \quad (5.6)$$

$$A(r) = \frac{p}{n} \left(\frac{M(P, c)}{M(T, c)} \log \frac{M(P, c)}{M(T, c)} + \frac{M(N, c)}{M(T, c)} \log \frac{M(N, c)}{M(T, c)} \right) \quad (5.7)$$

14-<SelectAnt> : :=1beam |2beam| 3beam| 4beam |5beam

<SelectAnt> : The fuzzy rule selection process, based on the values of the evaluation function and stopping criterion, selects a number of fuzzy candidate rules to enter the next iteration of the rule refinement process. As specified in this rule, the size of the beam can vary from 1 to 5. Regardless of the number of selected rules, the current best rule is replaced only if its evaluation function value is worse than the value of the best

selected rule.

15- $\langle \alpha - cover \rangle : := 0.5 \mid 0.6 \mid 0.7 \mid 0.8 \mid 0.9$

$\langle \alpha - cover \rangle$: During the rule selection phase, the parameter $\alpha - cover$ is used to decide the examples covered by a rule, it can be considered as a threshold. The values of this threshold are : 0.5, 0.6, 0.7, 0.8, 0.9.

16- $\langle Bayes \rangle : := \text{if (bayesSelector} \geq \langle \beta - belonging \rangle) \text{ formRule}$

$\langle Bayes \rangle$: The fuzzy refined rules are evaluated according to the fuzzy Bayes measure represented by bayesSelector terminal and it is defined as follows :

$$B(C_k|R) = \frac{\sum_{e \in T} \mu_{C_k}(e) \tau \mu_R(e)}{\sum_{e \in T} \mu_{C_k}(e)}$$

(5.8)

where τ is a t-norm such as minimum. If the rule strength is above a user-defined level $\beta - belonging$ then the rule is added to the fuzzy rule set, R is a fuzzy concept description, T is the training set, e is a soft instance and C is the set of concepts.

17- $\langle \beta - belonging \rangle : := 0.7 \mid 0.8 \mid 0.9 \mid 0.95$

$\langle \beta - belonging \rangle$: is a user-defined level to test If the rule strength is significant and to decide wether to add the rule to the fuzzy set or not. values of this threshold are : 0.7, 0.8, 0.9, 0.95.

18- $\langle \text{PrePrune} \rangle := (\text{Exlude1selector} \mid \text{Exlude2selector} \mid \text{ExcludeLastselector}) \langle \text{EvaluateAnt} \rangle$

$\langle \text{PrePrune} \rangle$: When this single fuzzy rule building process terminates, a new fuzzy rule is added to the fuzzy rule set being produced. But before this happens, a last operation can be performed as a result of applying rule 5 : fuzzy rule pre-pruning. Pre-Prune implements a pre-pruning method that tries to simplify a rule by removing one condition or a set of conditions from its antecedent. Fuzzy rules can be simplified in three ways : (1)- Removing one condition-at-a-time (Exlude1selector) from its antecedent, as long as the new rule is better than the original rule. (2)- Removing two conditions from the end of the rule antecedent (Exlude2selector), as long as the new rule is better than the original one or (3)- Removing the last added condition ($\text{ExcludeLastselector}$) from its antecedent.

19- $\langle \text{PostPrune} \rangle := \text{ExludeRule modelAcc} \mid \langle \text{ExludeSelectors} \rangle \text{ modelAcc}$

$\langle \text{PostPrune} \rangle$: Once the fuzzy rule set is completed, the fuzzy rule induction algorithm can still perform a last operation : post-pruning the fuzzy rule model. The presence of a post pruning step in the algorithm is determined by the application of NT1. Post-pruning methods can apply the same techniques used to pre-prune a single fuzzy rule to all the fuzzy rules in the rule model or by excluding a complete fuzzy rule from the fuzzy rule set.

5.5 Results and Discussion

Before we started testing the system we downloaded data sets that we used to train and test the system, we downloaded 10 data sets from [26,83], these data sets are from different public domains so that the system will not be tailored to a specific domain as we have mentioned before. All of these data sets have only numerical attributes, Table

```

1-<Start>                --><FuzzyRuleSet> [<PostPrune>]
2-<Fuzzy RuleSet>        --> For-each-concept <whileLoop> endFor.
3-<whileLoop>            -->while <Cond> <GenerateAnt> endwhile
4-<Cond>                  -->(Pos ≥ <α-cover>) ≠0 | ((Pos ≥ <α-cover>) ≠0 && (newAntExist))
5-<GenerateAnt>          --><InitializeAnt> <Secwhile> [<PrePrune>]
6-<InitializeAnt>        -->True | bestSEED | randomSEED
7-<Secwhile>              -->while <SecCond> <FindAnt> endwhile
8-<SecCond>               -->SelectorSetNotEmpty | SelectorSetNotEmpty && SelectorNeg <α-cover>
9-<FindAnt>               --><RefineAnt> <EvaluateAnt> <SelectAnt> |<RefineAnt> <Bayes> |
                        if numCovExp ( >1 <)(90%| 95%| 99%) then <RefineAnt> else <RefineAnt> |
                        if AntSizeSmaller (2| 3| 5| 7) then <RefineAnt> else <RefineAnt>
10-<RefineAnt>            --><IncludeSelectors>|<ExcludeSelectors>
11-<IncludeSelectors>     -->Include1selector | Include 2selectors
12-<ExcludeSelectors>    -->Exclude1selector | Exclude2selectors
13-<EvaluateAnt>         -->LaplaceEstimate | FuzzyLaplace| FuzzyLs-content| FuzzyAccFunction| FuzzyPurity
                        FuzzyInfGain| FuzzyEntropy
14-<SelectAnt>           -->1beam |2beam| 3beam| 4beam |5beam
15-<α-cover>              -->0.5 | 0.6| 0.7| 0.8| 0.9
16-<Bayes>               --> if ( bayesSelector ≥ <β-belonging>) formRule
17-<β-belonging>         -->0.7| 0.8| 0.9| 0.95
18-<PrePrune>            -->(Exclude1selector | Exclude2selector) <EvaluateAnt>
19-<PostPrune>           -->ExcludeRule modelAcc | <ExcludeSelectors> modelAcc

```

FIGURE 5.6 – The Fuzzy Set Covering Grammar

5.1 shows the data sets characteristics used in the experiments. The first column shows the number of classes of each set, the second one shows the type of attributes and the third one reports the domain of each data set.

	numClasses	Atts types	Domain
Iris	3	numerical	Biology
Pima	2	numerical	life sciences
Glass	7	numerical	physical sciences
Wine	3	numerical	physical sciences
Vehicle	4	numerical	imagery
Wbc	2	numerical	life sciences
Ecoli	8	numerical	life sciences
Ion	2	numerical	physical sciences
Puba	2	numerical	medical sciences
Haberman	2	numerical	medical sciences

TABLE 5.1 – Data Sets Description

5.5.1 Data Pre-processing

When the data set is in the crisp format, it can be written in attribute relation file format (arff). as we have already seen in the previous chapter. crisp data set have a structure as in the following example[84] (see Figure

```
@relation sport
@attribute outlook {sunny, cloudy, rainy}
@attribute temp real
@attribute humidity {humid, normal}
@attribute wind real
@attribute activity {volleyball, swimming, weights}

@data
sunny,    30,  humid,    26,  swimming ;1
sunny,    26,  normal,    5,  volleyball ;2
cloudy,    28,  normal,   12,  swimming ;3
cloudy,    23,  normal,   14,  volleyball ;4
rainy,    28,  normal,   20,  weights ;5
cloudy,    13,  humid,    24,  weights ;6
rainy,    10,  normal,   10,  weights ;7
cloudy,    12,  normal,   14,  volleyball ;8
sunny,    33,  humid,    22,  swimming ;9
sunny,    13,  normal,   33,  weights ;10
sunny,    31,  humid,     0,  swimming ;11
```

FIGURE 5.7 – A crisp data set (arff)

when using crisp data sets the rules will be written of the form : IF antecedent THEN consequent : $(\text{outlook} = \text{sunny} \wedge \text{temp} = 13 \wedge \text{temp} = 28) \rightarrow \text{weights}$

$\wedge$: Symbol of conjunction ; two expressions must be true simultaneously.

A set of values of each attributes is called an instance. Instances classified correctly under a given rule are called positive instances otherwise negative. When concept descriptions for a particular class attribute, suppose weights, are written all together it forms the rule for that particular class attribute.

In this part of our work we will need fuzzy data sets that are represented in files known

as Fuzzy attribute relation file format. When the same kind of data as in figure 5.7 is in fuzzy format it can be written in fuzzy attribute relation file format. It can be well understood using an example data set (see Figure 5.8).

```

@relation sport
@attribute outlook {sunny, cloudy, rainy}
@attribute temp {hot, mild, cold}
@attribute humidity {humid, normal}
@attribute wind {windy, calm}
@attribute activity {volleyball, swimming, weights}
@data
(.9 .1 .0), (1 .0 .0), (.8 .2), (.4 .6), (.0 .8 .2) ;1
(.8 .2 .0), (.6 .4 .0), (.0 1.), (.4 .6), (1 .7 .2) ;2
(.0 .7 .3), (.8 .2 .0), (.1 .9), (.2 .8), (.3 .6 .1) ;3
(.2 .7 .1), (.3 .7 .0), (.2 .8), (.3 .7), (.9 .1 .0) ;4
(.0 .1 .9), (.7 .3 .0), (.5 .5), (.5 .5), (.0 .0 1.) ;5
(.0 .7 .3), (.0 .3 .7), (.7 .3), (.4 .6), (.2 .0 .8) ;6

```

FIGURE 5.8 – A fuzzy data set (farff)

Rules generated by algorithms evolved by FuAGGE over farff data set are of the same format : IF antecedent THEN consequent, where the antecedent is a conjunction of antecedents, and the consequent is a linguistic term from the set of class variables. For example consider the rule IF sunny $\wedge$ cloudy $\wedge$ mild THEN weights 0.8

the number following the consequent is the value of truthfulness.

In order to be able to use the data in the downloaded set, we started fuzzifying these latters by using the following method [85], this was manually done and took a large amount of time. Yet it was possible to use Fuzzy Bexa to do it automatically and since we couldn't have access to Fuzzy Bexa we did it manually as it was done by [86].

The first step of data fuzzification is to fix the number of intervals of membership functions of each attribute then to choose the shape of the membership function. In

our experiments we decided to go with the triangular membership function, we should note that this choice is not based on any heuristic but rather based on the easiness of implementation and we chose 3 intervals for each attribute Label0, label1, label2. A step by step description of the algorithm is provided in figure 5.9. where b value is the

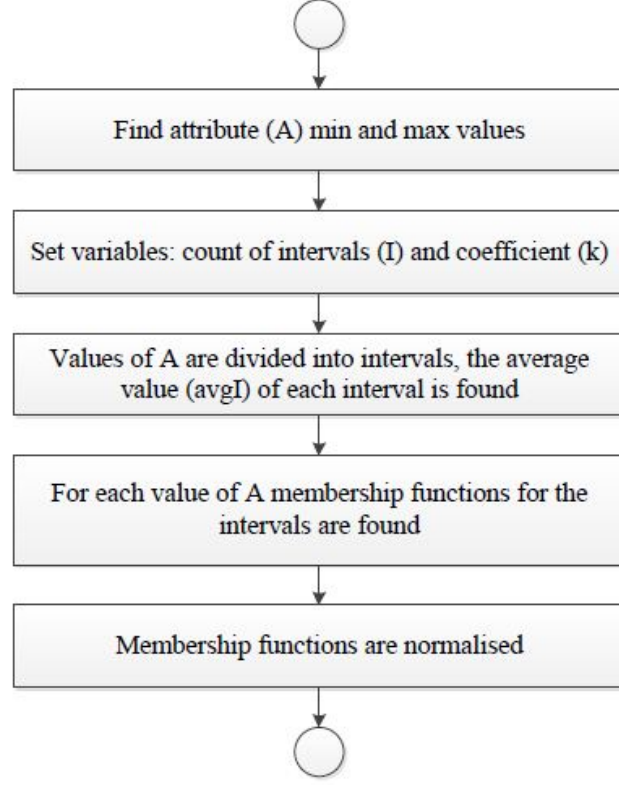


FIGURE 5.9 – Membership function construction algorithm

average value $avgA_n$, of the corresponding interval but the values a and c are calculated as follows :

$$avgA_{n-1} - kl$$

where $avgA_{n-1}$ is the average value of the preceding interval, k is a predefined coefficient $(0,1)$ and l is the length of the interval. Respectively :

$$c = avgA_{n+1} + kl$$

where $avgA_{n+1}$ is the average value of the next interval. Coefficient k was introduced to allow values to belong to more than two intervals. If $k=0$, every attribute value belongs to 1-2 intervals; after the introduction of k each value can belong to 1-3 intervals. The initial membership functions are calculated as follows :

$$\begin{cases} L(x) = \frac{x-a}{b-a}, a \leq x \leq b \\ R(x) = \frac{c-x}{c-b}, b \leq x \leq c \\ 0, \text{cits} \end{cases}$$

where $L(x)$ is the membership function of values that are to the left of the average interval value and $R(x)$ is the membership function of values that are to the right of the average interval value. In case where $avgA_{n-1}$ or $avgA_{n+1}$ does not have any value the intervals $avgA_{n-2}$ and $avgA_{n+2}$ are used. If the interval is an outer interval, values a or c take the values of the most outer value. After finding the initial membership functions, they are normalized resulting in functions that correspond to the following equation :

$$\sum_{i=1}^n \mu_s(x_i) = 1$$

This algorithm was applied to all attributes of all data sets used during experiments which resulted to farff files instead of arff ones. The data sets were divided into 2 groups, 60% of them (6) were used to train then to validate the models, and the rest were used for the purpose of testing. We should note that this division was done randomly. The meta-training set contains(Iris, Pima , Glass, Wine, Vehicle, WBC) and the meta-testing set contains (Ecoli, Ion, PUBA, Haberman), before starting the training phase each set of the meta-training set is sub-sampled using the 10-fold cross-validation in order to avoid over-fitting and to make predictions more generalizable as we have done in the previous chapter.

5.5.2 Experimentation and Discussion

The system needs 3 components to start the evolution process. The grammar which was introduced in section 5.3.4, the meta data sets and finally the grammatical evolution parameters, the number of generation was set to 60, the population size to 150, the mutation probability was set to 0.01, the crossover probability to 0.8, the selection method chosen is the tournament selection and we used the generational replacement. We should mention here that these parameters are not optimized but rather empirically chosen after analyzing a certain number of trials. In order to evaluate the newly generated fuzzy classifiers we computed the accuracies of 2 manually designed fuzzy rule based classifiers using all 10 data sets. The last column of Table 5.2 reports accuracies of the new generated fuzzy classifier (FuAGGE-classifier) while the remaining columns show accuracy values of the 2 manually designed classifiers (Furia, Fuzzy CN2), the upper 6 rows report the accuracies of the fuzzy rules sets generated by the FuAGGE-classifier and the 2 baseline ones using only the meta-training set (each row represents the test accuracy of a single set from the meta training set) these accuracy values are reported here to show the success of the training phase while the lower four rows of the table show the real predictive accuracy values because the FuAGGE-classifier has never met these sets during the training or validation phase. The aim of this part of the thesis was to extend the proposed system that uses the grammatical evolution method to automatically generate fuzzy rule based classifiers having an accuracy that can be at least competitive with the existing manually designed classifiers and also to compare the AGGE classifiers with its fuzzy version (FuAGGE). After we created the grammar and we prepared the data the implementation and the testing phase was launched, the system proved its ability to produce classifiers that are competitive with the manually designed ones. We can notice the performance of these former in Table 5.2, we should note that these accuracies were obtained by averaging the accuracy of the fuzzy rule

model generated by the FuAGGE-classifier for each test set over 10 iterations of the 10-fold-cross-validation method used during experiments, this also applies on the rest of the benchmark classifiers used for the purpose of comparison. It is worth mentioning that in Table 2 the new generated fuzzy classifier has practically the same results as the other methods and if we compare only the baseline methods with each others we can clearly notice that the Fuzzy CN2 records 5 wins over 1 for Furia even though the Furia is the more sophisticated one. It uses a growing, pruning and optimization phase just like its crisp version RIPPER and a fuzzy rule stretching method.

Now if we look at the FuAGGE-classifier accuracies we can notice how close are these accuracies to the baseline algorithms accuracies which is very interesting due to the fact that the FuAGGE-classifiers is automatically generated, and this removes a great deal of necessary time doing coding tasks. Humans designers can easily go wrong when parameterizing an algorithm during the design process, contrariwise the chance of having bad parameters when using automatic evolution of algorithms is very low. The last four rows show that the FuAGGE-classifier records 1 win against the baseline classifiers (Puba) and an equality with Fuzzy CN2 (Haberman), for the Ion data set the FuAGGE result was very close the best accuracy (90.38 versus 91.17) these results proved that the proposed approach can be very interesting.

However, the FuAGGE system is time consuming while evolving FuAGGE-classifiers and requires high computational power. Moreover, it can't be run properly under an ordinary computer, where the evolution process can take up to one week of continuous calculation, we should also note that this version of the system does not handle missing values and eliminates instances with missing value before using the data sets as we stated in the last chapter we should note also that this version uses only numeric attributes. Another drawback of this system is that the data is done manually which was a really hard and time consuming work. This should be fixed in the next version of the system.

	Furia	FuzzyCN2	FuAGGE Classifier
Iris	92.06	95.20	95.13
Pima	75.65	79.10	60.20
Glass	69.63	65.90	70.43
Wine	65.82	97.90	59.81
Vehicle	70.57	73.30	73.58
Wbc	95.28	98.11	96.26
Ecoli	83.63	83.90	79.82
Ion	91.17	89.5	90.38
Puba	67.83	57.44	70.10
Haberman	72.55	72.92	72.91

TABLE 5.2 – Accuracy rates (%) Using Both Meta-Sets

to test the choices made by the FuAGGE system in matter of terminals, we tested the frequencies of these latters during each generation as we have done with the AGGE system. Figure 5.10 illustrates the frequencies of the appearance of different rule refinement processes introduced by the grammar and these are : including one selector at a time to the antecedent of a rule (Include1Selector), including two selectors at a time to the antecedent of a rule (Include2Selectors), excluding one selector at a time from the antecedent of a rule (Exclude1Selector) and excluding two selectors at a time from the antecedent of a rule (Exclude2Selectors), we can clearly see that at the first generation choices are random and the symbols dominating the population start to appear during the evolution process and as we can see (Include1Selector) and (Exclude2Selectors) at the first generation (Initialization) were dominating the population until the generation 20 where things changed and the 3 terminals (Include1Selector, Include2Selectors, Exclude1Selector) started having closer frequencies and the terminal (Exclude2Selectors) started to disappear until it vanished by the generation 50, this is due to the random initialization in the first generation and to the elite choice during the new populations generation, we can also observe that the best refinements during the last 10 generations are (Include2Selectors, Exclude1Selector).

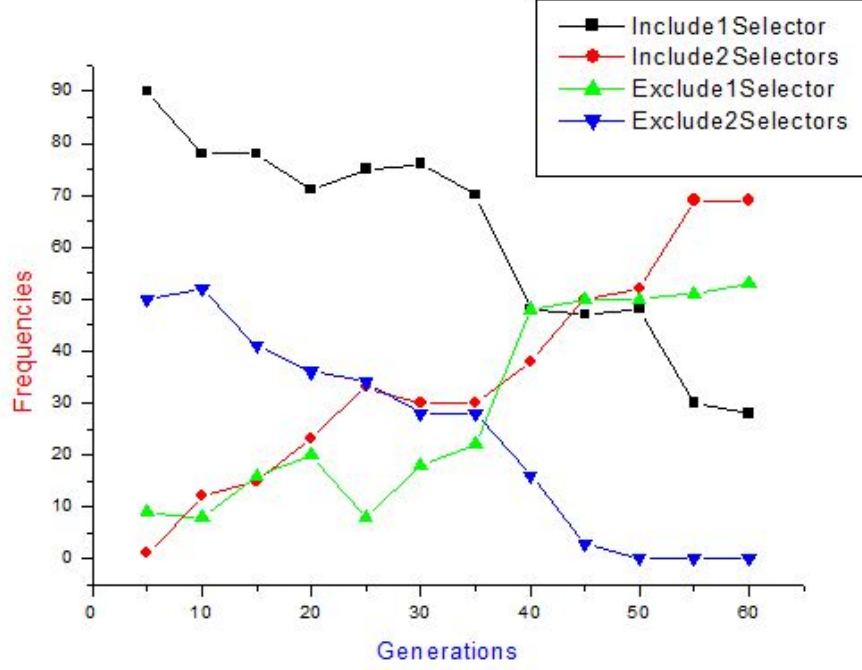


FIGURE 5.10 – Frequencies of Refinement Terminal Symbols During the 60 Generations

In figure 5.11 we can clearly see that the dominating rule evaluation measures by the end of generations are the (Laplace Estimate and the Fuzzy Accuracy Function) , the frequencies of these latters increases after each generation while for the (Fuzzy Laplace and Fuzzy Ls-Content) frequencies are kind of stable during generations and by the end they start to vanish. The remaining measure's frequencies greatly decrease during the evolution. It is worth mentioning that the accuracy and the Laplace functions often perform well and between the first three functions it is hard to decide the better one as they all perform very well even in the literature. The frequencies of the rest of functions decreased over generations but this doesn't mean that they can't perform good since some heuristics can perform better than other on some data domains and bad on others

Figure 5.12 presents the frequencies of terminals generated by the $\langle \alpha - cover \rangle$ Non-Terminal, we can clearly notice that the initialization phase the value 0.7 was dominating the generation and after the generation 15 things changed and the value 0.5 popped up and dominated the population. We can clearly see that the value 0.7 started decreasing

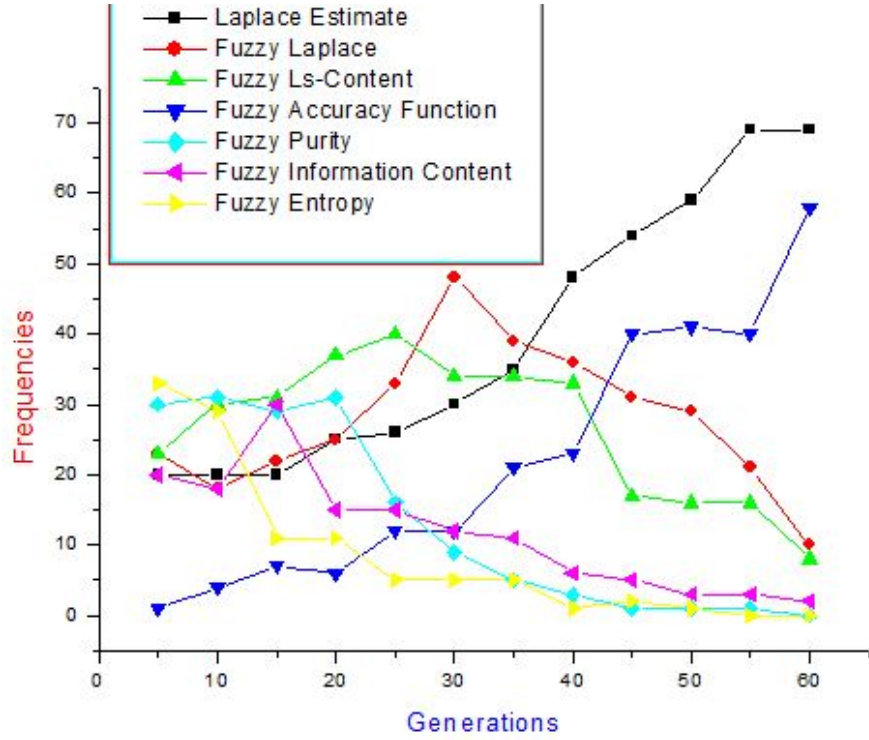


FIGURE 5.11 – Frequencies of Rules Evaluation Terminal Symbols During the 60 Generations

by the generation 20 and the values (0.8,0.9) completely vanished by the generations (15,35) respectively. We can easily conclude that the best values for the $\alpha - cover$ parameter are in the range [0.5-0.6].

We tested also different $\beta - belonging$ values during the 60 generations the results are represented in figure 5.13, we can notice in the graph that the distribution between the generations [20-35] balanced between the values (0.7,0.8,0.9) as for the value 0.95 it was absent during the first 10 generations because it hasn't been chosen during the initialization phase then it appeared by the generation 25 it was either due to a mutation or a crossover operations then it started vanishing again by the generation 45 , we can clearly notice that the best values are (0.8,0.9) whith the value 0.9 being the best value that could be chosen for the $\beta - belonging$ parameter.

To be able to work with real data it's needed to consider cognitive uncertainties. Because

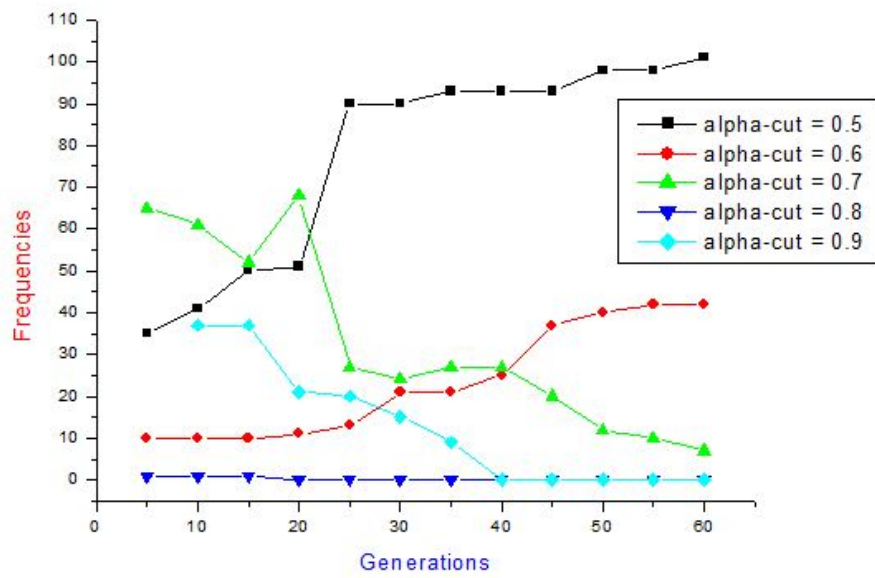


FIGURE 5.12 – Frequencies of Alpha-Covering Values During the 60 Generations

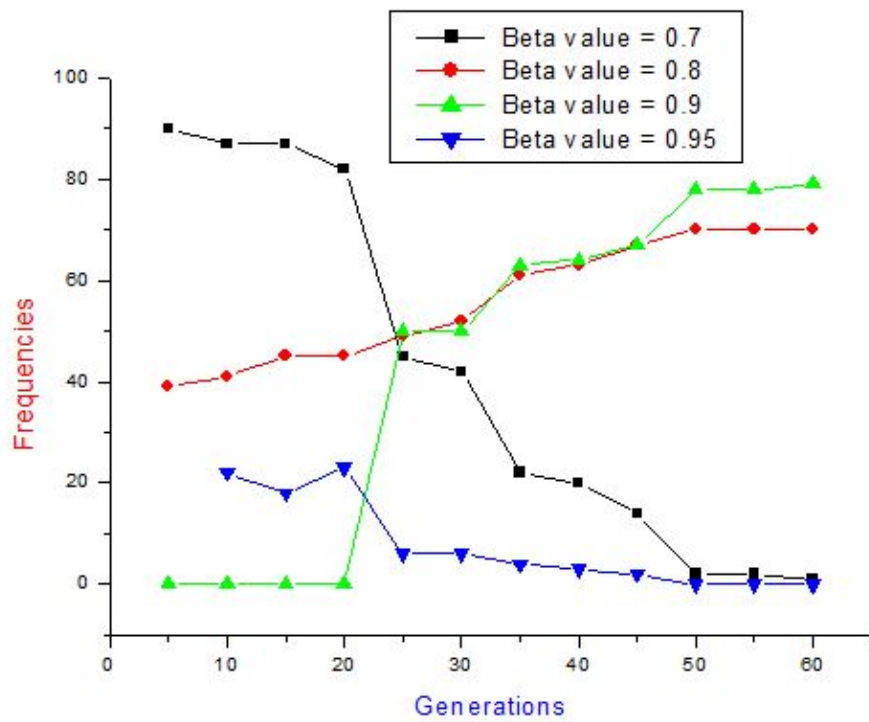


FIGURE 5.13 – Frequencies of the Beta-Belonging Values During the 60 Generations

of it, data aren't often accurate and contain some uncertainty. These uncertainties are caused by various reasons, such as : a) there is no reliable way how to measure something, e. g. a company's market value, a person's value, etc., b) it can be too expensive to measure something exactly, c) vagueness : it is associated with human's difficulty of making sharp boundaries, e. g. it's strange for people to say it's hot when it's 30 degrees (Celsius) and it's not hot when it's 29.9 degrees (Celsius), d) ambiguity : it is associated with one-to-many relations, i.e., situations with two or more alternatives such that the choice between them is left unspecified. These problems with real data have been successfully solved thanks to fuzzy sets and fuzzy logic for several years. Fuzzy sets are a generalization of (crisp) sets. In the case of a set, an element x either is the set's member or isn't. In the case of a fuzzy set, an element x is the sets member with given membership degree which is denoted by membership function that has a value in the continuous interval 0 to 1 [88]. Fuzzy sets are one of main ideas of Fuzzy Logic, which is considered as an extension of Boolean Logic as well as Multi-Valued Logic[87]. These are the main reason beyond choosing the fuzzy representation of rules and to extend the AGGE system to the FuAGGE Table 5.3 presents accuracies of classifiers evolved using our system (AGGE) in the first column and those evolved using (FuAGGE) in the second column. We should note that the parameters where set as follows : (generations=60, population size=150, crossover=0.8, mutation=0.01) for the AGGE system and (generations=60, population size=150, crossover=0.8, mutation=0.01) for the FuAGGE system. The upper 6 rows represent accuracy values using the meta training set and the lower ones using the meta testing sets. We can notice that the AGGE system recorded 4 wins versus 6 wins for the FuAGGE classifier this is because of the fuzzy nature of data and rules. the FuAGGE proved to be slightly more interesting than the AGGE in terms effectiveness of it's generated fuzzy classifiers. However we can't affirm this hypothesis until we compare both systems with a more large number of data sets.

	AGGE Classifier	FuAGGE Classifier
Iris	95.01	95.13
Pima	75.34	60.20
Glass	62.69	70.43
Wine	92.03	59.81
Vehicle	90.03	73.58
Wbc	94.93	96.26
Ecoli	73.31	79.82
Ion	88.14	90.38
Puba	76.44	70.10
Haberman	67.45	72.91

TABLE 5.3 – Accuracy rates (%) Using Both Meta-Sets (AGGE-classifiers versus FuAGGE-classifiers)

5.6 FuAGGE Classifiers Versus Manually Designed Classifiers

The purpose of this work was to propose a system that is able to automatically evolve fuzzy rule induction algorithms that are competitive with baseline ones. Results obtained after running and testing the system showed that the automatically evolved algorithms are highly competitive with manually designed ones.

In this section we will see the differences between the manually and automatically evolved ones. We will compare two algorithms that were automatically evolved by the FuAGGE system which performed well compared to baseline ones along with fuzzy PRISM algorithm.

Algorithm (5.14) represents the fuzzy PRISM pseudo code respecting the grammar's lexis for the sake of comparison, the PRISM algorithm generates an unordered set of fuzzy rules, it tries to produce rules that do not cover negative instance as quickly as possible, these rules may have very small coverage. Fuzzy PRISM follows for greedy search it starts with an empty antecedent and for each class it tries to build a set of

```

BEGIN:
  FruleSet= emptyset;
  FOReach concept Ci

    p=instances belonging to Ci;
    n=data-p;
    WHILE (not(p= 0))&&( alpha-belonging)  DO

      newAnt= empty;
      newAnt= addSelectors;
      1: FORall newAnt
        compute InfoGain;
      END;
      Select bestAnt;
      Compute B(Ci,bestAnt);
      IF (B > beta-belonging)

        create rule {if bestAnt then Ci};
        data=data-alpha-covered(rule,data);

      ELSE
        data2= instances alpha-cover(bestAnt);
        go to 1;
      ENDif;
    DONE;
  ENDfor;

END.

```

FIGURE 5.14 – Fuzzy-PRISM Pseudo Code

decision rules that do not cover any negative example, we can see that the fuzzy PRISM do not have a beam search as in the Fuzzy CN2 algorithm and it builds a rule by refining it using an operation that adds on a single antecedent at a time, it also uses the fuzzy information gain metric to evaluate the goodness of the generated rules. The next two algorithms (5.15 ,5.16) are fuzzy rule induction algorithms that were automatically evolved using the FuAGGE system which was trained using different data sets from different domains. As we can clearly see Algorithm 5.15 creates a random example rule as opposit to the fuzzy PRISM and fuzzyCN2 seen previously, then it refines the rule according to the number of examples covered by the rule if less than 90% it adds one condition at a time to the current rule else it removes two antecedents at a time then it evaluates the resulting refinements using the Laplace estimate metric and then it chooses the best two, if these latters have a Bayes greater than 0.7. Rules are created until no record is left in the data set. As we can see the difference between this algorithm and baseline ones is the initial rule type where the baseline one uses an empty rule and the generated one uses a random example, another difference is the refinement process where the new algorithm uses an if condition to choose between two refinement ways. When the rule set starts to be built, there is a lot of examples available in the data set, if number of the covered example of the current rule is less than 90% the algorithm tries to find the best combination of antecedents (best in terms of Laplace metric) else if the percentage is greater it removes two antecedents from the current antecedents set. another difference is also the beam used in the new generated algorithm that wasn't used in the fuzzy PRISM. We should notice that when using the FuAGGE generated algorithm the parameters alpha-covering and beta-belonging are automatically set and the user does not need to think about the best combination to use.

Algorithm 5.16 shows the pseudo code of a second algorithm generated by the FuAGGE system, this algorithm first starts the rule induction process by creating an empty rule

```

BEGIN:
  FruleSet= emptyset;
  FOReach concept Ci

    p=instances belonging to Ci;
    n=data-p;
    WHILE (not(p= 0))&&( alpha-belonging)  DO

      newAnt= randomSEED;
      IF (numCovExp < 90%) DO newAnt= Include1Selector;
      ELSE newAnt= Exlude2Selectors;

      1: FORall newAnt
        compute LaplaceEstimate;
      END;
      Select 2 bestAnt;
      Compute B(Ci,bestAnt);
      IF (B > beta-belonging(0.7))

        create rule {if bestAnt then Ci};
        data=data-alpha-covered(rule,data);

      ELSE
        data2= instances alpha-cover(bestAnt,0.5);
        go to 1;
      ENDif;
    DONE;
  ENDfor;

```

FIGURE 5.15 – Rule Induction Algorithm Generate Using FuAGGE system

and then according to the data covered it decides whether to add one or two conditions at a time after that the new refined rules is evaluated using the Lscontent to choose the best refinement after that a rule is refined, it tries to preprune it by adding either two antecedents if the number of covered examples is less that 95% or one antecedent then it evaluates the refined rules and chooses the best 4 as the best rules to add it to the

rule set. This process will be repeated until all data is covered.

BEGIN:

```
FruleSet= emptyset;
Foreach concept Ci

    p=instances belonging to Ci;
    n=data-p;
    WHILE (not(p= 0))&&( alpha-belonging) DO

        newAnt= TRUE;
        IF (numCovExp < 95%) DO newAnt= Include2Selectors;
        ELSE newAnt= Exlude1Selector;

        1: FORall newAnt
            compute LsContent;
        END;
        Select 4 bestAnt;
        Compute B(Ci,bestAnt);
        IF (B > beta-belonging(0.8))

            create rule {if bestAnt then Ci};
            data=data-alpha-covered(rule,data);

        ELSE
            data2= instances alpha-cover(bestAnt,0.6);
            go to 1;
        ENDif;
    DONE;
ENDfor;
```

FIGURE 5.16 – Rule Induction Algorithm Generate Using FuAGGE system

5.7 Summary

In this chapter we reviewed the extension of our new AGGE system. After reviewing the fuzzy set theory and the fuzzy set covering approach we reviewed components and

pseudo codes of some existing fuzzy rule induction algorithms.

We introduced in this chapter a new grammar that represent the overall structure of fuzzy rule based classifiers following the fuzzy set covering paradigm. We presented the new system extension (FuAGGE) along with the accuracy results obtained by the newly generated fuzzy rule set inducers. Due to the advantages that offer the fuzzy representation of knowledge our system generated fuzzy rules classifiers having interesting accuracies that outperformed crisp rules based classifiers, even though the grammar is a restricted one. Our next goal is to extend the grammar into a more detailed sophisticated one.

CHAPITRE 6

Conclusion

This thesis presented a new system called AGGE used for automatic generation of rule based classifiers using grammatical evolution. The AGGE system uses a grammar describing the overall structure of the rule based classifier, this grammar was inspired from the manually designed rule based classifiers. The grammar is used to introduce prior knowledge to the system and to bias the search.

The results presented in this thesis were obtained after performing extensive computational experiments, these results showed that the current system can be used to evolve rule induction algorithms that are competitive with the baseline ones. The evolved rule based classifiers showed to be robust and they can be applied on different data sets from different domains. The same system was used along with another grammar defining the structure of fuzzy rule based classifiers, the second experiments were applied to generate and produce fuzzy rule based classifiers. A grammar representing the basic structure of set a covering approach was introduced. The extension of the system was called FuAGGE, Accuracies of individuals evolved using the FuAGGE showed to be slightly better than baseline classifiers which is greater given that it was automatically designed.

The results of the computational experiments showed that the present system can effectively evolve rule based algorithms under two frameworks :

- The evolution of robust rule induction algorithms, which perform well in a variety of datasets used during the AGGE run

- The evolution of fuzzy rule induction algorithms, which perform well in a variety of datasets used during the FuAGGE run

The proposed method can be easily extended to a wide range of applications, provided that sufficient related data is available. The system can have the ability to automatically evolve crisp and fuzzy rule based induction algorithms that are tailored to a specific domain. The training data sets used in this kind of runs should belong to the same application domain consequently we will have different rule learners that belong the different data domains. This may open interesting opportunities and new trends in data mining and computational intelligence. For instance, our system could generate rule induction algorithms dedicated to medical data, or biological data, or physics engineering data. Algorithms would be parameterized in a way to better fit a specific domain and as a result, would achieve more accurate results in this particular field. An analysis of the evolved rule learners showed that the AGGE-classifiers and the FuAGGE-classifiers besides being competitive with manually designed one, many of them provides new ways of refining rules / pre-pruning and post pruning techniques. The new automatically evolved rule learner also have the tendency to be much simpler algorithms compared to some sophisticated manually designed rule induction algorithms.

In summary, this thesis has presented a new approach to automatically construct crisp and fuzzy rule based learners. These rule learners can be very promising when tailored to a particular application domain. In this context, the proposed method is a new and promising in the domain of algorithm construction. Although this thesis has presented an effective AGGE/FuAGGE system and reported experimental results, it still leaves room for several improvements and future research directions.

- Grammar Improvements : The grammar is the important element of the AGGE/FuAGGE system, it defines the search space and by improving and extending this grammar we will widen and change the search space, this may lead to new different

crisp/fuzzy rule learners. The grammr can be improved by adding new non-terminals dealing with either rule evaluation or rule refinements, it can also be improved by adding some components of some sophisticated rule induction algorithms such as (RIPPER, FURIA) that weren't taken in consideration in this version of the

- Fitness Function Improvements : In this version of the system we used a simple accuracy function to compute the fitness of possible solutions. The fitness can be improved by using multi objective fitness function such as using the accuracy along with the simplicity of the individuals
- Data Sets : data used also affects the mechanism of the search in a way , we can focus more on the data :
 - We can collect data with similar characteristic(sparse data, binary attributes...etc.), then we try to evolve for each group of data a learning algorithms
 - We can also extend the FuAGGE system by adding the data fuzzification phase to the system so that it will be done automatically by the system.

Bibliographie

- [1] S.N. Sivanandam, S.N. Deepa . *Introduction to Genetic Algorithms*, book, Springer Edition, 2008
- [2] H G. Beyer and H.P. Schwefel. *Evolution strategies : A comprehensive introduction*, In *Natural Computing*, Vol.1, 3-52, 2002
- [3] Jhon Koza. *Genetic programming : on the programming of computers by means of natural selection*, book, MIT press 1992
- [4] Gisele Lobo Pappa, Alex Freitas. *Automating the Design of Data Mining Algorithms*, book, Springer Berlin Heidelberg, 2010
- [5] R. McKay, Hoai, N.X., Whigham, P.A., Shan,Y. and O'Neill, M.
Grammar-based Genetic Programming : A review ,In *Genetic Programming and Evolvable Machines*, 365-396 2010
- [6] Man Leung Wong, Kwong Sak Leung . *Applying logic grammars to induce sub-functions in genetic programming* ,In *Evolutionary Computation, IEEE International Conference* , 737 - 740 vol.2 , 1995
- [7] O'Neill M., Hemberg E., Gilligan C., Bartley E., McDermott J., Brabazon A.
GEVA : Grammatical Evolution in Java, In *SIGEVolution ACM* , 17-23 vol.3, 2008
- [8] Norman Paterson, Mike Livesey . *Evolving caching algorithms in C by genetic programming* , In *Genetic Programming 1997 : Proceedings of the Second Annual Conference* , 262 - 267, 1997

- [9] Norman Paterson. *Genetic programming with context-sensitive grammars, Phd thesis, Saint Andrew's University, 2002*
- [10] Wong Man Leung , Leung Kwong Sak . *Data Mining Using Grammar-Based Genetic Programming and Applications ,book, Kluwer Academic Publishers, 2000*
- [11] Montana David J. *Strongly typed genetic programming ,In Evolutionary Computation Journal, vol 3, 199 - 230 , 1995*
- [12] R. McKay, Hoai, N.X., Whigham, P.A., Shan,Y. and O'Neill, M. *Grammar-based Genetic Programming : A review,In Genetic Programming and Evolvable Machines ,Vol 11 365-396, 2010*
- [13] McKay, R.I., Nguyen, X.H., Whigham, P.A. and Shan, Y. *Grammars in Genetic Programming : A Brief Review,In Progress in Intelligence Computation and Intelligence : Proceedings of the International Symposium on Intelligence, Computation and Applications Kang, L. Cai, Z. and Yan, Y. (eds), 3-18, 2005*
- [14] Adam Nohejl. *Grammar Based Genetic Programming, MSc Thesis , Charles University of Prague , 2011*
- [15] Adam Nohejl . *Grammatical Evolution, BSc Thesis, Charles University of Prague, 2009*
- [16] Alex A.Freitas . *Data mining and Knowledge Discovery with evolutionary algorithms, book , Springer Verlag , 2002*
- [17] Jeroen Eggermont.
Data Mining using Genetic Programming : Classification and Symbolic Regression, Phd Thesis, Leiden University, 2005
- [18] Liu Bing. *Web Data Mining : Exploring Hyperlinks, Contents, and Usage Data, book, Springer Edition, 2011*

- [19] Gisele Lobo Pappa. *Automatically Evolving Rule Induction Algorithms using Grammar-based Genetic Programming*, Phd Thesis, Kent University , 2007
- [20] Djazia Cendrowska . *PRISM : An Algorithm for Inducing Modular Rules*,In *International Journal of Machine Learning Studies*, 349 - 370 , 1987
- [21] Madara Gasparovica, Ludmila Aleksejeva, Valdis Gersons, Riga Technical University. *The Use of BEXA Family Algorithms in Bioinformatics Data Classification* ,In *Information Technology and Management Science* , 120-126 , 2012
- [22] Claude Sammut,Geoffrey I. Webb . *Encyclopedia of Machine Learning* ,book ,Springer edition , 2010
- [23] K M Ali, M J Pazzani. *HYDRA : a noise tolerant relational concept learning algorithm* ,In *13 Int. Joint Conference on Artificial Algorithms (IJCAI 93)* ,1064 - 1071 , 1993
- [24] Sholom Weiss, Nitin Indurkha. *Optimized Rule Induction*,In *IEEE Expert : Intelligent Systems and Their Applications*, vol 8 , 61 - 69 , 1993
- [25] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*, In *Studies in Computational Intelligence*. Springer, 2009
- [26] UCI Machine Learning Repository website url : <http://archive.ics.uci.edu/ml/>
- [27] Mihai Oltean , Crina Grosan. *A Comparison of Several Linear Genetic Programming Techniques* ,In *Complex Systems Journal*, vol 14 , 285- 313 , 2004
- [28] Wikipedia website url : <http://en.wikipedia.org/wiki/Context-free-grammar>
- [29] Peter Clark , Tim Niblett .*The CN2 Induction Algorithm*, In *Machine Learning*,261-283,1989

- [30] Ajith Abraham, Nadia Nedjah, Luiza De Macedo Mourelle. *Evolutionary Computation : from Genetic Algorithms to Genetic Programming ,In Studies in Computational Intelligence (SCI) vol 13, 1â20, 2006*
- [31] Alan Turing. *Intelligent Machinery Report ,National Physical Laboratory, 1948*
- [32] Richard M Friedberg. *A Learning Machine Part 1 ,IBM Journal of Research and Development, 1958*
- [33] Wolfgang Banzhaf, Frank D. Francone,Robert E. Keller, Peter Nordin *Genetic programming : an introduction : on the automatic evolution of computer programs and its applications ,book ,Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1998*
- [34] Alex Freitas. *A Review of Evolutionary Algorithms for Data Mining ,In Data Mining and Knowledge Discovery Handbook , 371-400, 2010*
- [35] Markus Brameier. *On Linear Genetic Programming ,Phd Thesis ,University of Dortmund, 2004*
- [36] Miller F Julian. *A Review of Evolutionary Algorithms for Data Mining ,In Natural Computing Series , 1-16, 2011*
- [37] Mihai Oltean. *Solving Even-Parity Problems using Traceless Genetic Programming ,In Proceedings of the 2004 IEEE Congress on Evolutionary Computation ,1813-1819, 2004*
- [38] Mihai Oltean. *Multi Expression Genetic Programming ,In International Conference on Computational Science, vol 2 ,670-673, 2004*
- [39] Candida Ferreira. *Gene Expression Programming : a New Adaptive Algorithm for Solving Problems. In CoRR cs.AI/0102027 , 2001*
- [40] Jason Brownlee. *Clever Algorithms : Nature-Inspired Programming Recipes, Book, , 2011*

- [41] Jacobus van Zyl. *Fuzzy set covering as a new paradigm for the induction of fuzzy classification rules. University of Mannheim , phd thesis, 2007.*
- [42] Yufei Yuan, Michael J. Shaw. *Induction of fuzzy decision trees, In Fuzzy Sets and Systems. Volume 69, Issue 2, pp. 125â139, 1995*
- [43] Nock, R. and Nielsen, F. *On Weighting Clustering, IEEE Trans. on Pattern Analysis and Machine Intelligence, 28 (8), 1â13, 2006*
- [44] Lin, C.-T., Lee, C. S. G. *Neural Fuzzy Systems : A Neuro-Fuzzy Synergism to Intelligent Systems. Upper Saddle River, NJ : Prentice Hall, 1996.*
- [45] Ching-Hung Wang, Chang-Jiun Tsai, Tzung-Pei Hong, Shian-Shyong Tseng. *Fuzzy Inductive Learning Strategies. Applied Intelligence, Volume 18, Issue 2, pp 179-193, 2003*
- [46] Jacobus Van Zyl , Ian Cloete. *FuzzConRI- A Fuzzy Conjunctive Rule. 2004*
- [47] Ian Cloete, Jacobus van Zyl. *Fuzzy rule induction in a set covering framework. IEEE T. Fuzzy Systems Volume 14, Issue 1, pp 93-110, 2006.*
- [48] Jens HÃ¼hn, Eyke HÃ¼llermeier. *FURIA : an algorithm for unordered fuzzy rule induction. Data Mining and Knowledge Discovery , Volume 19, Issue 3, pp 293-319, 2009.*
- [49] Gasparovica M., Aleksejeva L. *Using Fuzzy Algorithms for Modular Rules Induction. Scientific Journal of RTU. Volume 44 , pp 94-98, 2010.*
//
- [50] Ross Quinlan. *Learning Logical Definitions from Relations. In Machine Learning, vol 5(2), 239-266, 1990.*
- [51] H Jacobsson. *Rule extraction from recurrent neural networks : A taxonomy and review. In Neural Computation, vol 17, 1223-1263, 2005*

- [52] H Theron, I Cloete. *BEXA : A covering algorithm for learning propositional concept descriptions.* In *Machine Learning*, vol 24(1), 5-40, 1996
- [53] S M Weiss, N Indurkha. *Optimized rule induction.* *IEEE Expert : Intelligent Systems and Their Applications*, vol 8(6), 61-69, 1993
- [54] J Furnkranz, G Widmer. *Incremental reduced error pruning.* In *Proc. the 11th Int. Conf. on Machine Learning (ICML-94)*, 70-77, New Brunswick, NJ, 1994
- [55] Johannes Furnkranz. *Pruning Algorithms for Rule Learning.* In *Machine Learning*, vol 27(2), 139-172, 1997
- [56] Sago Deroski, Ivan Bratko. *Using the m-estimate in Inductive Logic Programming.* In *Machine Learning*, 1992
- [57] Johannes FÅ¼rnkranz . *Fossil : A Robust Relational Learner .* In *Proceedings of the European Conference on Machine Learning*, 122-137, 1994.
- [58] C A Brunk, M J Pazzani. *An investigation of noise-tolerant relational concept learning algorithms.* In L. Birnbaum and G. Collins, editors, *Proc. of the 8th International Workshop on Machine Learning*, 389-393. Morgan Kaufmann, 1991
- [59] William W Cohen. *Efficient Pruning Methods For Separate-And-Conquer Rule Learning Systems.* In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 988-994, 1993
- [60] Johannes FÅ¼rnkranz. *Pruning Methods for Rule Learning Algorithms.* In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, 321-336, 1994

- [61] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. *Meta-learning by landmarking various learning algorithms. In Proc. of the 1 Int. Conf. on Machine Learning, (ICML-OO), San Francisco, 743-750, 2000*
- [62] William H, Teukolsky Saul A, Vetterling William , Flannery B P. *Section 16.5. Support Vector Machines. In Numerical Recipes : The Art of Scientific Computing 3rd edition, Book, New York : Cambridge University Press, 2007*
- [63] Dominic S, Das R, Whitley D, Anderson C. *Genetic reinforcement learning for neural networks. In International Joint Conference on Neural Networks. Seattle, Washington, USA, vol 2, 71-76 1991*
- [64] John Georgen, Langley Pat. *Estimating Continuous Distributions in Bayesian Classifiers. In Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann, Montreal, Canada, 338â345, 1995*
- [65] Craig Stanfill, David Waltz. *Toward Memory-Based Reasoning. In Communications of the ACM, vol 29(12), 1213-1228 , 1986*
- [66] Breiman Leo. *Random Forests. In Machine Learning, vol 45(1), 5â32, 2001*
- [67] Ross Quinlan. *C4.5 : Programs for Machine Learning. In Morgan Kaufmann Publishers, 1993*
- [68] Ross Quinlan. *Induction of Decision Trees. In Mach. Learn. 1, vol 1, 81-106, 1986*
- [69] Tino Slamkovic . *Gramatical Evolution of technical Process. Phd Thesis, University of Zagreb ,2011*
- [70] R. Caruana, A. Niculescu-Mizil. *Data mining in metric space : an empirical analysis of supervised learning performance criteria. In Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD-04), 69-78, 2004*
- [71] Ian Dempsey, Michael O'Neill, Anthony Brabazon . *Evolving artificial neural networks. In Studies in Computational Intelligence, book, vol 194, 2009*

- [72] X. Yao. *Evolving artificial neural networks. In Proceedings of the IEEE, vol 87(9), 1423-1447, 1999*
- [73] Man L Wong. *An adaptive knowledge-acquisition system using generic genetic programming. Expert Systems with Applications, vol 15(1), 47-58, 1998*
- [74] S. B. Cho and K. Shimohara. *Modular neural networks evolved by genetic programming. In Proc. of the IEEE Int. Conf. on Evolutionary Computation, 681-684, 1996*
- [75] A. Suyama, N. Negishi, and T. Yamaguchi. *CAMLET : A platform for automatic composition of inductive learning systems using ontologies. In Proc. of the Pacific Rim Int. Conf. on Artificial Intelligence, 205-215, 1998*
- [76] H. Abe and T. Yamaguchi. . In R. Camacho and A. Srinivasan. *Comparing the parallel automatic composition of inductive applications with stacking methods. In Proc. of ECML/PKDD'03 - Workshop on Parallel and Distributed Computing for Machine Learning, 1-12, 2003.*
- [77] Sugeno, M. *Industrial applications of fuzzy control, Elsevier Science Pub. Co., 1985.*
- [78] Mamdani, E.H. *Applications of fuzzy logic to approximate reasoning using linguistic synthesis. IEEE Transactions on Computers, Volume 26, 12, pp. 1182-1191, 1977.*
- [79] Clark, P., Boswell, R. *Rule induction with CN2 : Some recent improvements. In : Proceedings of the Sixth European Working Session on Learning, pp. 151-163, 1991.*
- [80] P. Mart  n, F. J. Moreno-Velo *FuzzyCN2 : An Algorithm for Extracting Fuzzy Classification RuleLists. Proc. IEEE International Conference on Fuzzy Systems, Barcelona, Jul. 18-23, 2010.*
- [81] Jacobus van Zyl. *Fuzzy set covering as a new paradigm for the induction of fuzzy classification rules. University of Mannheim, PhD thesis, (2007).*

- [82] Xiaomeng Wang, Detlef D. Nauck, Martin Spott, Rudolf Kruse. *Intelligent data analysis with fuzzy decision trees*, *Soft Computing*, Vol. 11, Issue 5, 439–457, (2007).
 - [83] Hongbin Shen, Jie Yang, Shitong Wang, Xiaojun Liu. *Attribute weighted mercer kernel based fuzzy clustering algorithm for general non-spherical datasets*. In *Soft Computing*, Vol. 10, Issue 11, 1061-1073, (2006).
 - [84] Lin, C.-T., Lee, C. S. G. *Neural Fuzzy Systems : A Neuro-Fuzzy Synergism to Intelligent Systems*. Upper Saddle River, NJ : Prentice Hall, (1996).
 - [85] Ching-Hung Wang, Jau-Fu Liu, Tzung-Pei Hong, Shian-Shyong Tseng. *FILSMR : a fuzzy inductive learning strategy for modular rules*. *Proceedings of the Sixth IEEE International Conference On Fuzzy Systems*, Vol. 3, 1289-1294. Barcelona, (1997).
 - [86] Ching-Hung Wang, Chang-Jiun Tsai, Tzung-Pei Hong, Shian-Shyong Tseng. *Fuzzy Inductive Learning Strategies*. *Applied Intelligence*, Vol. 18, Issue 2, 179-193, (2003). .
 - [87] H Theron, I Cloete. *BEXA : A covering algorithm for learning propositional concept descriptions*. In *Machine Learning*, Vol. 24, Issue 1, 5-40, (1996). .
 - [88] Jens Huhn, Eyke Hullermeier. *FURIA : an algorithm for unordered fuzzy rule induction*. *Data Mining and Knowledge Discovery*, Vol. 19, Issue 3, pp 293-319, (2009).
- $\tilde{A}^1$
- [89] Li H., Wong L. *Knowledge Discovering in Corporate Securities Fraud by Using Grammar Based Genetic Programming*. *Journal of Computer and Communications*, Vol. 2, 148-156, (2014).
 - [90] De Silva A., Noorian F., Davis R., Leong P. *A Hybrid Feature Selection and Generation Algorithm for Electricity Load Prediction using Grammatical Evolution*,

12th International Conference on Machine Learning and Applications, Miami USA, 211-217, (2013).

- [91] Ming Dong and Ravi Kothari. *Classifiability Based Pruning of Decision Trees, Neural Networks. Proceedings IJCNN '01, International Joint Conference on Neural Networks (IJCNN), Washington, DC, (2001).*
- [92] Michelle Galea. *Fuzzy Rules from Ant-Inspired Computation. University of Edinburgh, PhD thesis, (2007).*
- [93] Vapnik, V. *Statistical Learning Theory. Wiley-Interscience (1998).*
- [94] Walter, D. and Mohan, C. K. ClaDia. *A Fuzzy Classifier System for Disease Diagnosis. In : Proceedings of the Congress on Evolutionary Computation, 1429-1435 (2000).*
- [95] Wang, C.-H., Liu, J.-F., Hong, T.-P. and Tseng, S.-S. *A Fuzzy Inductive Learning Strategy for Modular Rules. Fuzzy Sets and Systems, 103 :91-105 (1999).*
- [96] Wang, L.-X. and Mendel, J. M. *Generating Fuzzy Rules by Learning from Examples. IEEE Transactions on Systems, Man and Cybernetics, 22(6) :1414-1427 (1992).*
- [97] Wang, X., Chen, B., Qian, G. and Ye, F. *On the Optimization of Fuzzy Decision Trees. Fuzzy Sets and Systems, 112 :117-125 (2000).*
- [98] Wang, X. Z., Yeung, D. S. and Tsang, E. C. C. *A Comparative Study on Heuristic Algorithms for Generating Fuzzy Decision Trees. IEEE Transactions on Systems, Man and CyberneticsâB, 31(2) :215-226 (2001).*
- [99] Wang, Z. and Feng, B. E. . *Classification Rule Mining with an Improved Ant Colony Algorithm (2004).*
- [100] Whitley, D. *An Overview of Evolutionary Algorithms : Practical Issues and Common Pitfalls. Information and Software Technology, 43(14) :817-831 (2001).*

- [101] Witten, I. H. and Frank, E. Data Mining : Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2005).
- [102] Yang, L., Widyantoro, D. H., Ioerger, T. and Yen, J. An Entropy-Based Adaptive Genetic Algorithm for Learning Classification Rules. In : Proceedings of the Congress on Evolutionary Computation, 790-796 (2001).
- [103] Yuan, Y. F. and Shaw, M. J. Induction of Fuzzy Decision Trees. Fuzzy Sets and Systems, 69(2) :125-139 (1995).
- [104] Yuan, Y. F. and Zhuang, H. A Genetic Algorithm for Generating Fuzzy Classification Rules. Fuzzy Sets and Systems, 84(1) :1-19 (1996).
- [105] Zadeh, L. A. Fuzzy Sets. Information Control, 8 :338-3537
- [106] Janikow, C. Z. A Genetic Algorithm Method for Optimizing Fuzzy Decision Trees. Information Sciences, 89 :275-296 (1996).
- [107] Janikow, C. Z. Fuzzy Decision Trees : Issues and Methods. IEEE Transactions on Systems, Man and Cybernetics - Part B : Cybernetics, 28(1) :1-14 (1998).
- [108] Japkowicz, N. Learning from Imbalanced Data Sets : A Comparison of various Strategies. In : Learning from Imbalanced Data Sets : Papers from the AAAI Workshop, 10-15 (2000).
- [109] Japkowicz, N. Class Imbalances : Are we Focusing on the Right Issue? In : Proceedings of the Workshop on Learning from Imbalanced Datasets II, International Conference on Machine Learning (2003).
- [110] Jensen, R. and Shen, Q. Fuzzy-Rough Data Reduction with Ant Colony Optimization. Fuzzy Sets and Systems, 149 :5-20 (2005).

- [111] Jeong, J. and Oh, S.-Y. Automatic Rule Generation for Fuzzy Logic Controllers using Rule-Level Co-Evolution of Subpopulations. In : Proceedings of the 1999 Congress on Evolutionary Computation (CEC-1999), 2151-2156 (1999).
- [112] Kallel, L., Naudts, B. and Rogers, A. Theoretical Aspects of Evolutionary Computing. Springer (2001).
- [113] Kearns, M. A Bound on the Error of Cross-Validation using the Approximation and Estimation Rates. In : Advances in Neural Information Processing Systems, 183-189.
- [114] Gonzalez, A. and Perez, R. Completeness and Consistency Conditions for Learning Fuzzy Rules. Fuzzy Sets and Systems, 96 :37-51 (1998).
- [115] Nakashima, T., Ishibuchi, H. and Murata, T. Evolutionary Algorithms for Constructing Linguistic Rule-Based Systems for High-Dimensional Pattern Classification Problems. In : Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, 752-757 (1998).
- [116] Romao, W., Freitas, A. A. and Pacheco, R. C. S. A Genetic Algorithm for Discovering Interesting Fuzzy Prediction Rules : Applications to Science and Technology Data. In : Proceedings of the Genetic and Evolutionary Computation Conference, 343-350 (2002).
- [117] Walter, D. and Mohan, C. K. ClaDia : A Fuzzy Classifier System for Disease Diagnosis. In : Proceedings of the Congress on Evolutionary Computation, 1429-1435 (2000).
- [118] Yuan, Y. F. and Zhuang, H. A Genetic Algorithm for Generating Fuzzy Classification Rules. Fuzzy Sets and Systems, 84(1) :1-19 (1996).
- [119] Yang, L., Widyanoro, D. H., Ioerger, T. and Yen, J. An Entropy-Based Adaptive Genetic Algorithm for Learning Classification Rules. In : Proceedings of the Congress on Evolutionary Computation, 790-796 (2001).

- [120] <http://geneticprogramming.us/Selection.html>
- [121] Nevill-Manning, C.G.; Dept. of Comput. Sci., Waikato Univ., Hamilton, New Zealand; Holmes, G.; Witten, Ian H. The development of Holte's 1R classifier. In Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference (1995).

Appendices

ANNEXE A

A Note on the Fu/AGGE System's Execution Time

In this part we discuss the factors which tend to define the best and worst case scenarios in a run of the proposed AGGE/FuAGGE system instead of performing formal analyses of the computational time and time complexity. Theoretical studies are not very often performed when using evolutionary algorithms. If we analyse the behaviour of the system AGGE/FuAGGE we can easily conclude that the most time consuming task in the AGGE/FuAGGE system is the evaluation of the individuals. Since each individual represents a complete rule induction algorithm and it should be tested on a variety of data sets. We can say that the runtime of the fitness function of the AGGE/FuAGGE is the most time consuming part of the system and it relies on three elements :

- The number of data sets in the meta-training set.
- The size of each data set : number of attributes (and attributes values, in the case of nominal attributes) and examples in each data set in the meta-training set.
- The rule induction components used by the AGGE/FuAGGE to produce a complete rule induction algorithm.

This third element, has a great impact on the rule induction algorithms runtimes. This is because, when analyzing the symbols of the grammar, we notice that a few of them perform operations with a time complexity higher than linear. Add2, for instance, considers all the combinations of attribute/value pairs two by two, so its time complexity is quadratic with respect to the number of attributes.

We report here the best/worst runtimes empirically obtained for each of the experiments with the AGGE/FuAGGE. All the experiments were performed on I7 processor machines with 16 Go RAM and running Windows. Table [A.1](#) shows the results of the best and worst case scenarios when running the proposed AGGE/FuAGGE with different numbers of data sets in the meta-training set. The first column presents the type of the system used (AGGE or FuAGGE) used during the experiments, followed by the number of data sets in the meta-training set and the best and worst run times, reported in the format hours : minutes. It is important to remark that the execution time of the rule induction algorithms can still be improved, and their code is not optimized.

System	number Data-sets	best Time	worst time
AGGE	4	25 :53	36 :37
AGGE	8	29 :54	32 :06
AGGE	10	25 :34	34 :21
AGGE	13	59 :31	70 :36
FuAGGE	3	9 :17	20 :54
FuAGGE	4	16 :38	18 :09
FuAGGE	6	21 :50	32 :40

TABLE A.1 – AGGE/FuGGE runtime for different experiments configurations