Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time

Thesis submitted by Jaume Bacardit i Peñarroya
in partial fulfillment of the requirements for
the degree of Doctor in Computer Science

Thesis advisor : Dr. Josep Maria Garrell i Guiu
Computer Science Department
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull

Barcelona, October 18, 2004

Abstract

Pittsburgh genetic-based machine learning (DeJong, Spears, & Gordon, 1993) is, among others (Wilson, 1995; Venturini, 1993), an application of evolutionary computation techniques (Holland, 1975; Goldberg, 1989a) to machine learning tasks. The systems belonging to this approach are characterized by evolving individuals that are compete rule-sets, usually variable-length. Therefore, the solution proposed by these kind of systems is the best individual of the population.

When using this approach, we have to deal with some problematic issues such as controlling the size of the individuals in the population, applying the correct degree of generalization pressure across a broad range of datasets, reducing the considerable run-time of the system, being able to solve datasets with diverse kind of attributes, etc. All these issues become even more critical when applied to modern-day data mining problems.

In this thesis we have the general objective of adapting the Pittsburgh model to handle successfully these kind of datasets. This general objective is split in three: (1) Improving the generalization capacity of the model, (2) Reducing the run-time of the system and (3) Proposing representations for real-valued attributes. These three objectives have been achieved by a combination of four types of proposals, some of them focused only on a single objective, some others solving partially more than one objective at the same time. All these proposals are integrated in a system, called *GAssist* (Genetic clASSIfier sySTem).

An experimentation process including a wide range of data mining problems based on many different criteria has been performed. The experiments reported in the thesis are split in two parts. The first part studies several alternatives integrated in the framework of *GAssist* for each kind of proposal. The analysis of these results leads us to propose a small number of global configurations of the system, which are compared in the second part of the experimentation to a wide range of learning systems, showing how this system has competent performance and generates very reduced and interpretable solutions.

Resum

L'enfocament de Pittsburgh (DeJong, Spears, & Gordon, 1993) de l'aprenentatge evolutiu és, entre d'altres alternatives (Wilson, 1995; Venturini, 1993), l'aplicació de les tècniques de computació evolutiva (Holland, 1975; Goldberg, 1989a) a l'aprenentatge artificial. Els sistemes que apliquen aquest enfocament es caracteritzen per fer evolucionar individus consistents en un conjunt de regles, normalment de mida variable. Per tant, la solució proposada per aquests sistemes és el millor individu de la població.

Quan es fa servir aquest enfocament, cal tractar amb alguns assumptes com el control de la mida dels individus de la població, l'aplicacio del grau correcte de pressió de generalització per un espectre ampli de problemes, la reducció del temps de càlcul del sistema, tractar problmes amb diversos tipus d'atributs, etc. Tots aquests problemes esdevenen encara més crítics quan es preten solucionar problemes de mineria de dades.

L'objectiu general d'aquesta tesi és adaptar l'enfocament de Pittsburgh de l'aprenentatge evolutiu per tal de solucionar amb èxit aquest tipus de problemes. Aquest objectiu general es divideix en tres parts: (1) Millorar la capacitat de generalització, (2) reduir el cost computacional del sistema i (3) proposar representacions per atributs reals. Aquests tres objectius s'han assolit mitjançant una combinació de quatre tipus de contribucions. Algunes d'aquestes propostes sols solucionen un dels objectius. D'altres en poden solucionar més d'un al mateix temps. Totes aquestes propostes estan integrades en un únic sistema anomenat *GAssist* (Genetic clASSIfier sySTem).

L'experimentació realitzada inclou un ampli ventall de problemes de mineria de dades. Aquesta experimentació s'ha dividit en dues parts. En la primera part s'ha experimentat amb diverses alternatives per separat per cada un dels quatre tipus de contribucions fetes en la tesi. L'objectiu d'aquesta part és la de poder proposar un subconjunt reduït de configuracions del sistema que podem considerar que tenen un bon rendiment en general. En la segona part de l'experimentació de la tesi aquest conjunt de configuracions bones s'ha comparat a un ampli ventall de sistemes d'aprenentatge, fent servir diversos tipus de representacions del coneixement, de tècnica d'aprenentatge, etc. Aquests experiments mostren com el sistema *GAssist* té un rendiment competitiu i genera solucions compactes i altament interpretables.

Resumen

El enfoque de Pittsburgh (DeJong, Spears, & Gordon, 1993) del aprendizaje evolutivo es, entre otras alternativas (Wilson, 1995; Venturini, 1993), la aplicación de las técnicas de computación evolutiva (Holland, 1975; Goldberg, 1989a) a las tareas de aprendizaje artificial. Los sistemas que aplican este enfoque se caracterizan por hacer evolucionar individuos que consisten en un conjunto de reglas, habitualmente de tamaño variable. Por lo tanto, la solucion propuesta al problema a resolver por este tipo de sistemas es el mejor individuo de la población.

Cuando se usa este enfoque es necesario solucionar correctamente algunos asuntos como el control del tamaño de los individuos de la población, aplicar el grado correcto de presión de generalización sobre un conjunto amplio de problemas, reducción del coste computacional del sistema, tratar problemas con tipos de atributo diversos, etc. Todos estos problemas son todavía más serios cuando se pretende solucionar problemas modernos de minería de datos.

El objectivo general de esta tesis es adaptar el enfoque de Pittsburgh para solucionar con éxito este tipo de problemas. Este objetivo general se divide en tres partes: (1) mejorar la capacidad de generalización, (2) reducir el coste computacional y (3) representaciones para atributos reales. Estos tres objetivos se han logrado mediante la combinación de cuatro tipos de contribuciones. Algunas de estas propuestas sólo solucionan uno de les objetivos. Otras pueden solucionar más de un objetivo al mismo tiempo. Todas estas propuestas están integradas en el sistema llamado *GAssist* (Genetic clASSIfier sySTem).

La experimentación realizada incluye un amplio espectro de problemas de minería de datos. Esta experimentación está dividida en dos partes. En la primera parte se ha experimentado con diversas alternativas por separado para cada uno de los cuatro tipo de contribuciones realizadas en esta tesis. El objetivo de esta parte de la experimentación es poder proponer un subconjunto reducido de configuraciones del sistema que podamos considerar como "buenas" en general. En la segunda parte de la experimentación de la tesis este conjunto de configuraciones buenas ha sido comparado a diversos sistemas de aprendizaje artificial que representan diversos tipos de representaciones del conocimiento, de técnicas de aprendizaje, etc. Estos experimentos muestran como el sistema *GAssist* tiene un rendimiento competitivo y genera soluciones compactas y altamente interpretables.

Acknowledgements

This thesis is the result of four years of hard work, and I would like to thank many people for helping me during this long journey. First of all I would like to thank Josep Maria, my advisor, for his advise, for accepting me as a PhD student, and for giving me the resources I have used to develop my work.

I also would like to thank Xevi and Maria for their constant support, advise, friendship, and for the huge amount of red ink used to correct my papers, it really helped improving them. Many thanks go for all the people with whom I have shared the development of this thesis in my department, your friendly support and all the fun during lunch time helped me a lot: Carlos (Vallespí), Mireya, David, Xavi, Carlos (Villavieja), Rosa, Agustín, Alex, and Tona. Many thanks to the rest of the research group.

I would like to thank my family. My parents Jaume and Montserrat, my siblings Maria, Jordi, Montse and Mercè. You always had confidence in me regardless of how weird did it sound everything I was doing. An special message goes to Jordi and Montse: I would not like to be the only PhD in the family so I wish that you succeed in the PhD adventure you both have started recently. I also would like to thank my friends Joan, Ivan, Xavi, Pilu, Raül, Judith, Jesús, Àlex, Marta, and many others.

I will always remember with fondness the great months I spent at IlliGAL, in the University of Illinois at Urbana-Champaign. Many thanks to Professor Goldberg for accepting my visit and for all the meetings and advise received, and also to Kumara, Hussein, Martin, Tian-Li, Chen-Ju, Ying-Ping, Kei, Feipeng, Nao, Claudio and Pier Luca. Thanks for your advice, your friendship and for all the fussball games. I am also grateful to the spanish community in the university, Javier, Neus, Carmen, Felix, Ana, Pablo, Hector and many others. You made me feel at home.

Finally, I would have been unable to obtain this PhD degree without funding. I acknowledge the help provided by the Department of Universities, Research and Information Society (DURSI) of the Autonomous Government of Catalonia under grant 2001FI 00514, the support provied by the Health Research Fund of the Spanish Health Ministry under grant FIS 00/0033-02 and the the Spanish Research Agency (CICYT) under grant TIC 2002-04036-C05-03.

Contents

List of Tables

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

1.1 Framework

Artificial intelligence (AI), broadly defined, is concerned with intelligent behavior in artifacts. Intelligent behavior, in turn, involves perception, reasoning, learning, communicating and acting in complex environments. AI has as one of its long-term goals the development of machines that can do these things as well as humans can, or possibly ever better (Nilsson, 1998).

This thesis is focused in the area of AI called *machine learning* (ML). ML deals with the question of how to construct programs that automatically improve with experience (Mitchell, 1997). In recent years the number of tasks and specific problems that can be handled with the techniques belonging to this discipline has risen. Some examples of these tasks are prediction, decision-support systems, scheduling, automatic classification, etc.

What happens if the experience used by this program to learn starts growing? Can the standard learning techniques extract useful information from huge volumes of information? Can the learning process be performed in a reasonable time? The answer to these questions is a broad discipline called *data mining* (Witten & Frank, 2000), which includes several kinds of techniques to preprocess information, extract knowledge from this information (which can be done with adapted *ML* techniques) and analyze or post-process the extracted knowledge. This thesis, as its name indicates, has as its aim the development of ML techniques that can be used in data mining tasks.

Moreover, the focus of the thesis is one of the sub-categories of ML: *supervised learning*, which is defined as the learning process where there is some kind of tutor (automatic or human) that gives the learner direct feedback about the appropriateness of its performance. This is usually achieved by providing the learning system with a *training set*, experience which has

been labeled with the correct response to it, so that the learning system can adjust itself to behave correctly.

More specifically, this thesis is focused in a paradigm of supervised ML called *evolutionary learning*, or *genetic-based machine learning* (GBML). This paradigm can be defined as any kind of learning task which employs as its search engine a technique belonging to the evolutionary computation (Michalewicz, 1996) field. Evolutionary computation (EC) techniques are optimization [1] tools inspired loosely in certain biological processes like the *Darwinian* natural selection or the genetic codification of life forms. Typically, a population of candidate solutions (individuals) are transformed (evolved) through a certain number of iterations of a cycle containing an almost blind recombination of the information contained in the individuals and a selection stage that directs the search towards the individuals considered *good* by a given evaluation function.

Traditionally, there are two approaches of GBML reported in the literature, called *Michigan approach* and *Pittsburgh approach*. De Jong did a brief general description (De Jong, 1988) of these two models:

> "To anyone who has read Holland (Holland, 1975), a natural way to proceed is to represent an entire rule set as a string (an individual), maintain a population of candidate rule sets, and use selection and genetic operators to produce new generations of rule sets. Historically, this was the approach taken by De Jong and his students while at the University of Pittsburgh (Smith, 1980; Smith, 1983), which gave rise to the phrase *the Pitt approach*.
>
> However, during the same time period, Holland developed a model of cognition (classifiers systems) in which the members of the population are individual rules and a rule set is represented by the entire population (Holland & Reitman, 1978; Booker, 1982). This quickly became known as *the Michigan approach* and initiated a friendly but provocative series of discussions concerning the strengths and weaknesses of the two approaches."

Thus, there is one approach, the *Pittsburgh*, very close to the essence of EC techniques, where an individual is a complete solution, and there is a competition between the candidate solutions in the population, and the search space exploration is made using almost blind (without knowledge domain) genetic operators. In short, a very simple learning paradigm.

On the other hand, the *Michigan* approach deals with individuals that are only one part of the final solution, which is the whole population. Also, the individuals cooperate in the

[1] Although, like in this case, they can be applied to other kind of tasks beside optimization, like search, learning or scheduling

population, instead of compete. Furthermore, some kind of reinforcement learning mechanism is needed to identify and promote the good individuals and EC techniques are only used, from time to time, to explore the search space. In short, this is a model with a much more complex structure than the mentioned above.

These two approaches represent two very different ways of interpreting the contribution of evolutionary computation to ML. The *Pittsburgh* model is an optimization tool applied to learning tasks that uses an EC technique as its main driving force. The *Michigan* model has been designed specifically for learning and it is a combination of several modules, one of them being some EC method.

In recent years there has been much more work reported in the literature on *Michigan* systems than on *Pittsburgh* ones. What is the reason for this? The *Pittsburgh* model has some problems and open questions which are difficult to answer, although some of them also affect the *Michigan* approach:

- The size of the solutions. As said above, an individual encodes a complete solution to the learning task. This usually means a set of rules. If this set of rules has fixed size, some criterion (difficult to set *a priori*) is needed to set its size. If the individual encodes a variable-length set of rules, it has to deal with a problem identified as *bloat effect* (Soule & Foster, 1998), which consist in the growth without control of the size of the individuals.

- The run time. Pittsburgh systems have had for a long time the reputation of being *very slow* systems. Evaluating an individual means classifying all instances in the training set, and the cost of classifying an instance depends on the size of the individual. Thus, if there is no strong control over this size or if the size of the training set is high (as in data mining tasks), the run-time problem of this model gets even worse.

- The generalization capacity. The main problem of using an optimization tool to perform learning is to manage the system to learn the concept represented by the training set, instead of learning the training set itself, that is, to achieve good generalization capacity. Even if the fitness function is adjusted to cope to some extent with this problem, it is difficult to automatically adjust the system to perform well over a broad range of domains.

- Representations for real-valued attributes. Most of the traditional GBML systems only handle nominal attributes. Nowadays, and especially for data mining tasks, it is a basic requirement to handle real-valued attributes

The objective of this thesis is to answer these questions.

1.2 Objectives and contributions of this thesis

The contributions presented in this thesis are an answer (but obviously, not the only one) to these open questions of the *Pittsburgh* model applied to data mining. The title of the thesis ("Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization and run-time") defines the general objectives of the thesis:

- Reducing the run-time of the system
- Improving the generalization capacity of the *Pittsburgh* model
- Proposing representations for real-valued attributes

Four kinds of contributions, that correspond to the four central chapters of this thesis, have been made:

Explicit and static default rule In the encoding used by most of the knowledge representations in *GAssist*, the set of rules contained as an individual is interpreted as a decision list (Rivest, 1987) (an ordered set of rules). If we apply this strategy in the evolutionary framework, often the system evolves emergently a default rule. Default rules can be very useful in combination with a decision list because the size of the rule set can be reduced significantly. With a smaller rule set, the search space is reduced resulting in two potential advantages: (1) the learner has to learn less rules (representing only the other classes of the dataset) and (2) with a smaller rule set the system may be less sensitive to over-learning, potentially increasing the test accuracy of the system. However, performance of the system is strongly tied to the learning system choosing the correct class for this default rule. This thesis reports the research done on extending the knowledge representation used in GAssist with an explicit and static default rule, and the policies studied to choose the correct default class.

Adaptive Discretization Intervals knowledge representation The contributions of this thesis to the area of representations for real-valued attributes is a knowledge representation called *adaptive discretization intervals (ADI) rule representation*. The approach chosen to handle these attributes is through a discretization process, but in a special way: the intervals used in the rules are created by joining together some adjacent cut-points proposed by a discretization algorithm. Also several discretization algorithms are used at the same time, letting the system choose the most suitable one for each dataset. With these two characteristics, the proposed representation gains robustness and has an efficient exploration of the search space.

Windowing techniques for generalization and run-time reduction With the objective of reducing the run-time of the system, some windowing techniques, that use only a subset of the training examples to perform the fitness computations, were studied and tested. The unexpected observation extracted from these tests was that one of these techniques, called *Incremental Learning with Alternating Strata (ILAS)* also generated extra generalization pressure. Thus, in this thesis this study on windowing techniques has been extended with a double objective: (1) tuning the windowing techniques to maximize the accuracy performance of the system and (2) achieving the maximum run-time reduction possible while maintaining the accuracy of the non-windowed system. Also, specific strategies have been proposed and tested to deal with small and large datasets. Moreover, some theoretical models have been developed that can partially predict the behavior of the system.

Bloat control and generalization pressure methods As stated above, bloat control and generalization pressure are very important issues in the design of Pittsburgh GBML systems, in order to achieve simple and accurate solutions in a reasonable time. The bloat control deals with a problem, identified as *bloat effect*, related with the unlimited growth of the size of the individuals. The same techniques used to control bloat if properly adjusted and combined with other techniques can be helpful to introduce generalization pressure into the system, evolving more accurate but compact solutions potentially having better test accuracy. A side effect of applying this pressure towards short individuals is a run-time reduction always desirable in this context. Therefore, several techniques have been studied in this context.

Thus, we have one contribution completely focused on the generalization pressure: the default rule mechanisms, one contribution completely focused on representations for real-valued attributes: the *ADI* representation, one contribution designed for run-time reduction, but that also introduces generalization pressure: the windowing mechanism and finally one set of contributions designed to control the size of the individuals to avoid the bloat effect and to apply generalization pressure, but with the side effect of some run-time reduction. The three objectives of the thesis are achieved by combining methods of the four kinds of contributions.

These contributions are all integrated into a single system, called *GAssist* (Genetic clASSIfier sySTem). The system was born (Bacardit & Garrell, 2002d) as a simple reimplementation of the *GABL* system (DeJong & Spears, 1991), one of the classical references of the *Pittsburgh* approach, but through the years (Bacardit & Garrell, 2002c; Bacardit & Garrell, 2002a; Bacardit & Garrell, 2002b; Bacardit & Garrell, 2003d; Bacardit & Garrell, 2003c; Bacardit & Garrell, 2003a; Bacardit & Garrell, 2004; Aguilar, Bacardit, & Divina, 2004; Bacardit, Gold-

berg, & Butz, 2004; Bacardit, Goldberg, Butz, Llorá, & Garrell, 2004) it has been extended gradually to include all the contributions of this thesis, thus creating a competent learning system for data mining tasks.

1.3 Road Map

The thesis has been structured in three parts. The first part contains an overview of required background material to draw the context where this thesis is placed. This background material has been split in two chapters. Chapter 2 is focused on machine learning topics. It is not an extensive nor complete review of the machine learning field, because its aim is to describe only the topics closely related to this thesis. It starts with some general machine learning definitions, paradigms and knowledge representations and then focuses on the specific topics: rule induction, discretization algorithms, scaling-up techniques and handling missing values.

Next, chapter 3 focuses specifically on evolutionary computation and genetic-based machine learning (GBML). It starts with a description of general evolutionary computation topics and then it focuses on GBML. First with a description of the main GBML approaches and then focusing on specific topics: representations for real-valued attributes, scaling-up of GBML systems and control of the bloat effect.

The second and central part of the thesis describes the contributions to the Pittsburgh GBML model presented. It has six chapters. Chapter 4 focuses on the experimental framework of the thesis. This means two parts: defining the basic mechanism of *GAssist* that cannot be considered novel contributions and defining the test design used for the experimentation in this thesis. The test design includes the datasets chosen, the experimental methodology of the tests, and the kind of statistical tests used to analyze the results of these tests.

The next four chapters describe the four kinds of contributions made in this thesis. Chapter 5 is focused on the explicit default rule mechanisms. After illustrating the motivation of these mechanisms, a complete definition of the changes made to the knowledge representation are presented, together with the basic default class determination policies. Next, some more sophisticated policies are introduced, and all the alternative options are tested.

Chapter 6 is focused on the *ADI* knowledge representation for real-valued attributes. The chapter first defines the representation and all its basic operators. A brief study of the dynamics of this basic version motivate the addition of another operator. Next, the experimentation starts by testing each candidate discretization algorithm alone in the framework of ADI. The results of these tests motivate the proposal of several groups of discretization algorithms, that are tested in several settings to determine which is the best set of discretization algorithms

and the best conditions to use it.

Chapter 7 describes the contributions done to windowing techniques for generalization and run-time reduction. The chapter starts with a description of the development process that led to the proposal of the *ILAS* technique used in the rest of the chapter. Next, there is a description of the models proposed to predict the behavior of *ILAS*. The experimentation with *ILAS* is split in two parts, one for small datasets and the other for large ones. Specific strategies of the use of *ILAS* are proposed for each kind of tests.

Chapter 8 describes the contributions on bloat effect and explicit generalization pressure mechanisms. After illustrating the need of this control, the basic mechanism used to control the bloat effect is presented and analyzed. Next, the two alternative mechanisms studied to apply extra generalization pressure are described. Finally, the combinations of these techniques are tested in several different scenarios.

The experimentation reported in these four chapters has been exclusively inside the framework of *GAssist*. Chapter 9 contains an extensive comparison of *GAssist*, using all the previous four kind of contributions, against several well-known machine learning systems that represent a broad range of knowledge representations and learning mechanisms. Several conclusions are extracted from these tests about the strong and weak points of the system used in this thesis.

The third part of the thesis contains the conclusions and further work. Chapter 10 contains the conclusions. First it summarizes the conclusions extracted from each of the four kinds of contributions and then, based on these partial conclusions and the results of chapter 9, some general conclusions are proposed. A similar structure is used for the further work in chapter 11.

The fourth part of the thesis are the appendixes which contain the full results of the experimentation reported in this thesis.

Part I

Background material

Chapter 2

Machine learning and rule induction

This chapter presents a general description of the field of application of this thesis: Machine learning. This chapter does not pretend to be an exhaustive review of the machine learning topic, but seeks to provide enough background material to be able to place and relate the contributions contained in this thesis within the machine learning field. For this reason, most of the chapter will focus on explaining the topics that are closest to our field of application: rule induction systems, representations for real-valued attributes and discretization techniques, scaling-up techniques and handling of missing values.

The chapter is structured as follows. First, section 2.1 will provide a brief definition of what machine learning is , and what its main paradigms are. Next, section 2.2 will focus specifically on the machine learning task we are dealing with in this thesis, the classification problem, by defining it and the concepts that are going to be used in the rest of the thesis. Section 2.3 will describe the main knowledge representations (and the corresponding inference mechanisms) used to solve the classification problem, focusing especially on rules, the knowledge representation investigated in this thesis. Section 2.4 will show some types of learning algorithms used to generate sets of rules. The next three sections will describe how some specific topics closely related to this thesis are handled in the machine learning field. These topics are the discretization process and discretization algorithms in section 2.5, the scaling-up of machine learning systems, or, how can we handle a large volume of information, in section 2.6 and how we can deal with missing values in section 2.7. Finally, section 2.8 provides a summary of the chapter.

2.1 Machine learning and its paradigms

The field of machine learning is concerned with the question of how to construct programs that automatically improve with experience (Mitchell, 1997). This field draws on concepts and results from many fields, including statistics, other paradigms of artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory, among others. Moreover, Mitchell defines the machine learning process as:

Definition 1 *A computer program is said to* **learn** *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Depending on how E, P and T are defined, we can label the paradigms or families of paradigms of the machine learning field. Three examples of the application of this formalism follow:

A checkers learning problem :

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: practice games against itself

A handwriting recognition learning problem :

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

A robot driving learning problem :

- Task T: driving on public four-lane highways using video sensors
- Performance measure P: average distance traveled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

There are many ways to classify the machine learning paradigms. One of them (Langley, 1995) classifies the learning paradigms depending on *how* do they learn, defining five categories:

Inductive learning This paradigm employs condition-action rules, decision trees or similar logical knowledge structures. Information about classes or predictions are stored in the action sides of the rules or the leaves of the tree. Learning algorithms in the rule-induction framework usually carry out a greedy search through the space of decision trees or rule sets, using statistical evaluation functions to select attributes to incorporate into the knowledge structure.

Instance-based or case-based learning This paradigm represents knowledge in terms of specific cases or experiences and relies on flexible matching methods to retrieve these cases and apply them to new situations. One common approach simply finds the stored case nearest (according to some distance metric) to the current situation, then uses it for classification or prediction.

Analytic learning This paradigm also represents knowledge as rules in logical form but typically employs a performance system that uses search to solve multi-step problems. A common technique is to represent knowledge as inference rules, then to phrase problems as theorems and to search for proofs. Learning mechanisms in this framework use background knowledge to construct proofs or explanations of experience, then compile the proofs into more complex rules that can solve similar problems either with less search or in a single step.

Connexionist learning This paradigm, also called *neural networks*, represents knowledge as a multilayer network of threshold units that spreads activation from input nodes through internal units to output nodes. Weights on the links determine how much activation is passed on in each case. The activations of output nodes can be translated into numeric predictions or discrete decision about the class of the input.

Evolutionary learning This paradigm, as stated previous in the introduction chapter of this thesis, is defined as any kind of learning task which employs as its search engine a technique belonging to the evolutionary computation (Michalewicz, 1996) field. Evolutionary computation techiques are optimization tools inspired loosely in certain biological processes like Darwinian natural selection or the genetic codification of life forms. Tipically, a population of candidate solutions (individuals) are transformed (evolved) through a certain number of iterations of a cycle containing an almost blind recombination of the information contained in the individuals and a selection stage that directs the search towards the individuals considered *good* by a given evaluation function. A broader description of evolutionary computation and evolutionary learning (also known as genetic-based machine learning) can be found in chapter 3.

The rationale of this clasification is more historical than scientifical. Actually, we can consider that the subset of evolutionary learning techniques which deal with rule sets can also be labeled as inductive learning, because they generate rules, although the transformation mechanisms of the candidate solutions (as will be seen in chapter 3 are less directed than in "regular" induction systems.

Another classification, using a more general point of view suggests three main categories:

Supervised learning It is defined as the learning process where there is some kind of tutor (automatic or human) that gives the learner direct feedback about the apropriateness of its performance. Relating this definition to the formal machine learning definition, we must have the performance measure P to perform supervised learning.

Unsupervised learning This kind of learning is caracterized by having no performance feedback, that is, no P. In this case, the task of the learning system is to construct some kind of knowledge based only on the flow of experience E, typically trying to identify the regularities existing on E.

Reinforcement learning This paradigm could be considered a middle point of the two previous ones. In this case, the feedback acts in a subtle way, indicating the performance of the system as a kind of *reward*, good or bad, instead of informing in a specific way *what* is being done correctly of incorrectly.

In this thesis we are dealing exclusively with supervised learning. Therefore, the rest of this chapter will be focused only on this general paradigm.

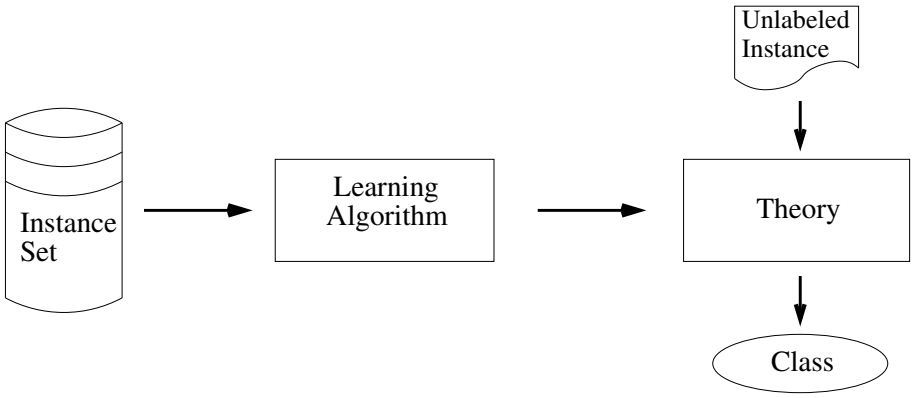
2.2 The classification problem

This thesis deals with a kind of supervised learning task called **classification**. Webster's dictionary defines classification as "the act of forming into a class or classes; a distribution into groups, as classes, orders, families, etc., according to some common relations or affinites. In our case, the classification process can be formally defined as:

Definition 2 *Given a set of instances $\mathcal{I} = \{i_1, \cdots, i_n\}$, each of them labeled with a finite set of classes $\mathcal{C} = \{c_1, \cdots, c_m\}$, the task of classification is to create a certain theory $\mathcal{T}$ based on $\mathcal{I}$ and $\mathcal{C}$ that, given an unlabeled new instance, can give a prediction of the class of this instance*

Graphically, the full classification process is represented in figure 2.1. This is a representation of the two stages of the life cycle of a learning cycle: *training* and *exploitation*. However, when

Figure 2.1: Representation of the learning process for classification tasks



we are developing and studyng learning systems, we have to simulate the exploitation stage. This simulation is done by splitting our set of labeled examples into two non-overlapping sets: the *training set* and the *test* set.

The test set is used to validate that the generated theory is correct, that is, that the learning system has been able to model the concept represented by the instances in the training set, instead of modelling only the instances themselves. If the learning system has been able to generate a correct model, when we try to classify the instances in the test set, the rate of instances for which we are able to predict its class correctly [1] should be equal or only slightly lower than the accuracy that the theory obtains in the training set.

Definition 3 *The capacity of generating a theory that models correctly the concept or concepts represented by the training set is known as* **generalization capacity**. *A good performance on the test set is a sign of generalization.*

The instances that are processed by the learning algorithm have a regular form: each instance contain a finite and fixed set of elements, called **attributes**. An attribute is a feature that characterizes the instance. We can have several types of attributes, but we usually deal with only three of them:

nominal attributes are the attributes that can take a value from a finite and fixed set

integer attributes are the ones that have a numeric value of type integer, usually with predefined upper and lower bounds. They may be treated as nominal attributes

[1] The rate of correctly classified instances is known as **accuracy**

real-valued attributes are the attributes that take a numerical value of any kind, with no restrictions

As described previously, each instance has another element: an associated class. The class can be considered as a nominal attribute, it can only take a value from a discrete and finite set of values. Sometimes, some of the attributes of an instance are undefined. This is what we define as **missing values**, and it is a very important problem, because it can distort the generation of the theory, and affect the generalization capacity of the learning system. Section 2.7 is focused on techniques to deal with this problem.

Another problem affecting the classification task is **noise**. We assume that the labels of all our training instances are correct, but this might not be true. The causes can be several, such as errors in the knowledge adquisition or processing, but the important point is that if we have wrongly labeled instances in our training set, the generation of a theory can be distorted. Therefore, the generalization capacity diminishes. Handling noise correctly is an important feature for a learning algorithm.

2.3 Knowledge representations and inference mechanisms

In the previous section we defined the task of classification as the construction of a theory that models the concept or concepts represented by a set of examples. This section deals with how this theory is, that is, what knowledge representation we use to construct it.

Definition 4 *The object of* **knowledge representation** *is to express knowledge in computer-tractable form, such that it can be used to help agents perform well. (Russel & Norvig, 1995)*

In this section we will describe some relevant knowledge representations, especially focusing on rule sets, the representation used in the contributions presented in this thesis. All of these representations are currently being used. A newcomer to the field may ask why there is not a clear winner, are all representations equally good? The answer to this question is a concept called **representation bias**. Each knowledge representation language restricts the space of possible solutions because of the limitations of its definition (Langley, 1995). This concept is very related to another one, the **inductive bias**.

Definition 5 *Consider a concept learning algorithm L for the set of instances X. Let c be an arbitrary concept defined over X, and let $D_c = \{\langle x, c(x)\rangle\}$ be an arbitrary set of training examples of c. Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after*

training on the data D_c. The **inductive bias** *of L is any minimal set of assertions B that affect the classification of any input instance (Mitchell, 1997).*

That is, the kind of theory that we can generate and the kind of predictions that we can made are affected by the chosen representation language and the chosen learning algorithm that builds the theory based on the representation language. This is a fact that affects any learning algorithm. Can this be considered a negative effect? No, because these introduced bias make feasible the task of learning, but it means that any learning algorithm and knowledge representation can be only the best algorithm in a certain subset of problems. This concept is usually adressed as the **selective superiority problem** (Brodley, 1993).

The knowledge representations described in the following subsections are:

- Rule sets
- Decision trees
- Sets of instances
- Bayes networks
- Artificial neural Networks

This list can be somewhat confusing, because the name of the knowledge representation is the same as the learning algorithm applied to it. The aim of this chapter is to focus on the machine learning issues related to the rest of the thesis. Therefore, we will only show the minimum details of the learning algorithms of all knowledge representations except the rule sets, which is our interest. The next section will focus on learning algorithms for rule sets.

2.3.1 Rule sets

Rule sets are the most ancient knowledge representations, and probably the easiest to understand. Their origins can be tracked back to the ancient Greek philosophers and they propositional logic. A rule set is a finite set of entities which are labeled rules. Rules can take very different forms, and also there are many different ways to interpret the rule set in order to classify an input instance. These two issues are described as follows:

Syntax of a rule

There are many ways to define a rule. From a general point of view, a classification rule (also known as if-then rule) has the following form:

If $condition$ Then $action$

Usually the condition of a rule is a predicate in a certain logic, and the action is an associated class, meaning that we predict $action$ for an input instance that makes true $condition$. Most rule syntax used in learning systems can be reduced to this general form, although the process might not be completely direct. One example of this is *inductive logic programming* (Lavrac & Dzeroski, 1993), a class of inductive learning systems that deal with Horn clauses.

Moreover, a typical definition of this condition is a conjuntion of terms, each of them related to an attribute of the input instance. Some examples of these terms, for nominal and real-valued attributes follow:

- $attribute_i$ is equal to $value_i^j$
- $attribute_i$ is equal to $value_i^{j_1}$ or $value_i^{j_2}$
- $attribute_i$ is irrelevant
- $attribute_i$ belongs to the interval $[low, high]$
- $attribute_i$ is lower than $value$
- $attribute_i$ is higher than $value$

Classification process

If we have a set of rules, and we are classifying an input instance, it can happen that there are more than one rules tha tare true for this instance. In this moment we have to use some kind of mechanism to decide which rule will be used or how to combine the outcome of each matched rule to produce a prediction. There are several ways to solve it. Three typical ways are described as follows:

- Decision lists. One of the ways to decide which rule is chosen to classify an input instance is to define previously an ordering or hierarchy of rules. Within this ordering, the first rule that is true for the input instance will be used to predict its class. This structure is usually known as **decision list** (Rivest, 1987).
- Heuristics based on the previous performance of each rule. If the rule has been used previously, we know, among other metrics, how accurate it is, how general it is (how often it has been used). Based on these metrics and others, we can constuct some formulas to rank the rules and choose the rule in conflict with more ranking based on the desired formula. A large summary of these heuristic formulas can be found in (Fürnkranz, 1999). Two of these formulas are shown in figure 2.2.

Figure 2.2: Heuristic formulas to choose the best rule

Accuracy The simplest approach is to examine the past experience of the rule and compute its accuracy:

$$Accuracy = \frac{C}{T}$$

Where C is the number of correctly classified instances and T the total number of instances matched by the rule.

Laplace accuracy The previous heuristic does not take into account the number instances covered. This can lead to promote rules that cover very few examples although with high accuracy, which can lead to a loss of generalization capacity. The laplace accuracy tries to fix this problem by introduce into the accuracy formula a term that allows some degree of mis-classifications, if the rule is used frequently:

$$\text{Laplace Accuracy} = \frac{C+1}{T+NC}$$

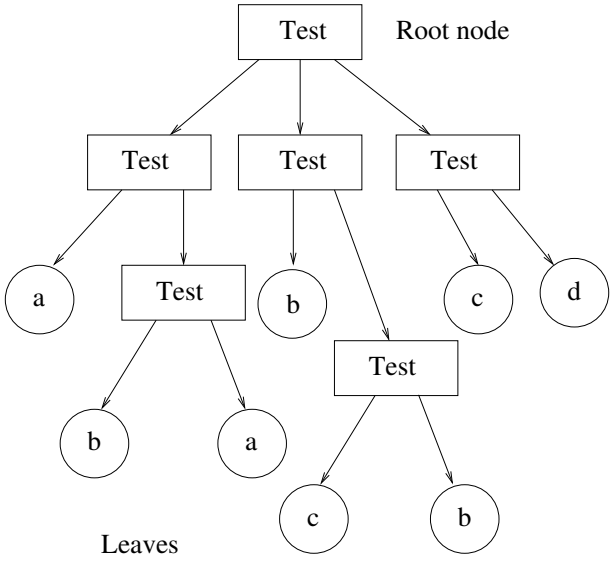
Where NC is the number of classes in the dataset. This formula approximates the previous accuracy one when the rule is highly covered, but tends to a very low accuracy $(1/C)$ when it is used very infrequently.

- Voting process. As an alternative to choosing a single rule to classify an instance, we can combine the outcome of all the rules that match it. As usual there are several alternatives. The simplest one is to choose the majority class from the matched rules. Another alternative is to sum, for each class, the number of instances of the class matches previously by these rules, and then choose the class most covered. This schema is used in *CN2* (Clark & Boswell, 1991).

2.3.2 Decision trees

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some *attribute* of the instance, and each branch descending from that node corresponds to one of the possible values/range of values for this attribute (Mitchell, 1997). A graphical display of this knowledge representation is shown in figure 2.3.

Figure 2.3: Representation of a decision tree

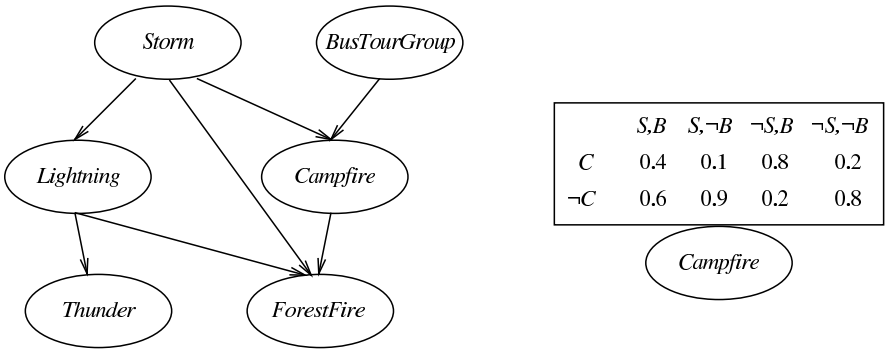


A decision tree classify an input instance by performing a number of tests, starting from the root node and following a path in the tree until a leaf node is found. The class prediction is the label of the leaf. A test of a node of the tree can be named *univariate* if it affects only one attribute of the instance (like in the above definition) or *multivariate* if it affects more than one (or all) of the attributes of the instance. For real-valued attributes, among other options, the tests can take the form of a relational operator ($value < \theta$,$value > \theta$,$value \in [lower, upper]$) in the univariate way, or a lineal combination of the attribute values in the multivariate way. These two alternatives are also known as **ortogonal decision trees** and **oblique decision trees**, exemplified by the *C4.5* (Quinlan, 1993) and *CART* (Breiman, Friedman, Olshen, & Stone, 1984) systems, respectively.

2.3.3 Set of instances

This knowledge representation consists of storing a set of instances, either taken from previous experience of syntetic ones, and classifies input examples, in general, looking for the k instances from the stored set that are nearest to the input example, based on some distance metric. When we have selected the k closest instances to this example, the outcome prediction can be based on a simple voting mechanism, or other more sophisticated techniques. There are several distance functions, a good review of them can be found in (Wilson & Martinez,

Figure 2.4: Representation of a bayes network



2000). Maybe the most common one is the *Minkowsky* distance:

$$D(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r} \tag{2.1}$$

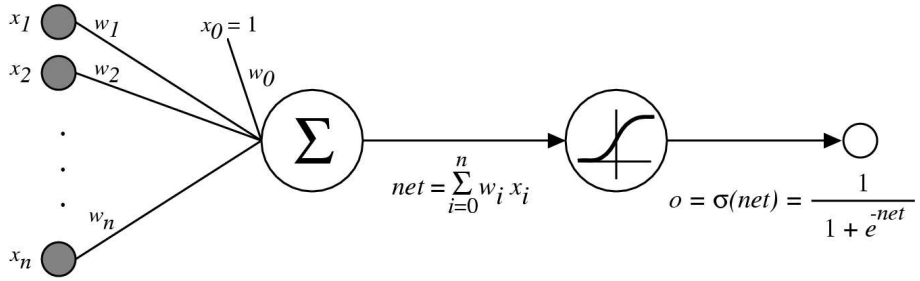
With this knowledge representation several questions remain opened to the learning system: How do we initialize the set of instances? Do we add or substract instances over the time? Most of these questions are handled in the Case-Based Reasoning (Aamodt & Plaza, 1994) field.

2.3.4 Bayes networks

This knowledge representation is an example of a learning-related technique inspired by an external field: statistics, and specifically the Bayes theorem. A bayes network (Pearl, 1988) is a directed acyclic graph where each node represents a random variable. The arrows conecting nodes define a dependency relation: the node origin of the arrow influence the pointed node. These influences are quantified by conditional probabilities. Each variable (node) influenced by another node has a conditional probability table associated to it. If not, it has an associated table containing the marginal distribution of the random variable. We can see a bayes network as the graphical representation of a joint probability distribution of the attributes in the domain, as shown in figure 2.4 (Mitchell, 1997).

There are many inference mechanisms working on bayes networks, but usually we have one node associated to each possible class of the domain. We have to compute the probability of all of these nodes and select the one with highest probability. Because this variable depends on other variables, we will need to first compute the probability distribution of these variables.

Figure 2.5: Representation of a perceptron



Once we have them, we can apply the bayes theorem to compute this probability. This process will be performed backwards for each node until we arrive at a node that depends on nothing. The probability of this node will be computed from its marginal probability table and the input instance. There are several algorithms used to construct bayes networks, a very simple but powerful one is *Naive Bayes* (Langley, Iba, & Thompson, 1992).

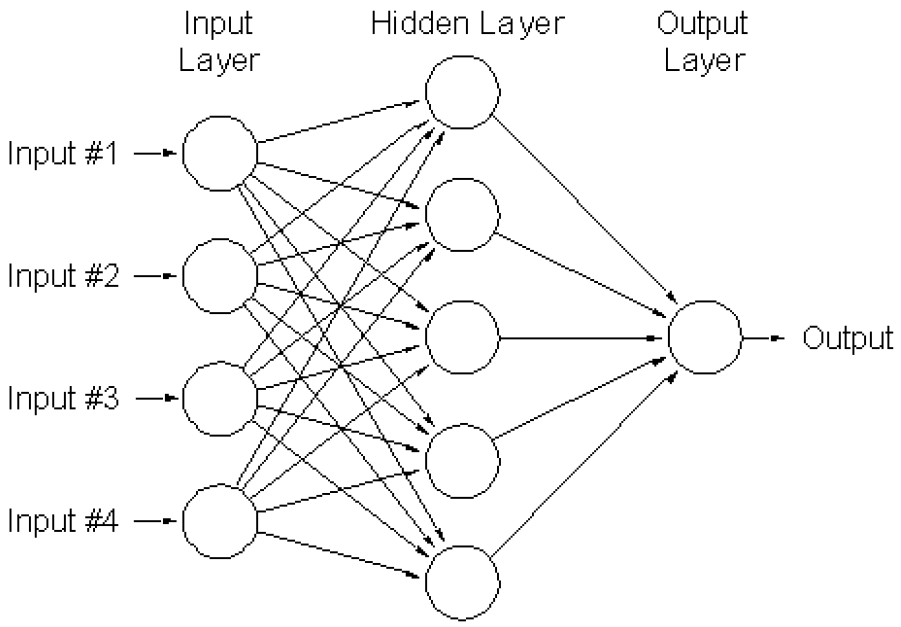
2.3.5 Artificial neural networks

The study of artificial neural networks has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons. As a rough analogy, artificial neural networks are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to many other units) (Mitchell, 1997).

Artifical neural networks is one of the oldest areas of study in the artifical intelligence field (McCulloch & Pitts, 1943), applied to several different types of problem like character, voice and face recognition (LeCun, Boser, Denker, Henderson, Howard, Hubbard, & Jackel, 1989; Lang, Hinton, & Waibel, 1990; Cottrell, 1990), or learning to drive (Pomerleau, 1993). As stated above, a neural network is an interconnection of several small and simple processing elements, inspired in the neurons. One of the most common of these "artificial neurons" is called a *perceptron*. In short, a perceptron receives the input of several other units and performs a weighted sum of these input values. This sum is the input of an *activation function* that decides the output of the perceptron. Figure 2.5 (Mitchell, 1997) shows the representation of a perceptron.

A perceptron can only solve linearly separable problems. To be able to handle more complex problems, it needs to be connected into a network. A very common interconnection topology is

Figure 2.6: Representation of a multi-layer perceptron



called *multi-layer perceptron* (MLP), because the perceptrons are organized in structured layers, as represented in figure 2.6. The figure also shows how all connections among perceptrons go to the next layer, in what is called a *feed-fordward network*. For a classification problem, each neuron labeled input would receive an attribute of the instance to classify, and we would fetch the prediction from the output layer. If we have binary classification problems with positive/negative examples, a single output perceptron is enough. For problems with more possible outcomes we need a perceptron for each class in the dataset.

The learning process in such networks consists of adjusting the weights of each perceptron, after having decided the number of neurons in the hidden layer. A very common weight-adjusting algorithm is called *backpropagation*. This algorithm employs a gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs (Mitchell, 1997). It starts by adjusting the weights of the perceptrons in the output layer and then continues with the previous hidden layer and so on. This is the reason of the algorithm name, because it adjust the weights from output to input.

Figure 2.7: The separate-and-conquer meta learning system

Separate-and-conquer algorithm
Input : $Examples$
$Theory = \emptyset$
While $Examples \neq \emptyset$
    $Rule$ = FindBestRule($Examples$)
    $Covered$ = Cover($Rule$,$Examples$)
    If RuleStoppingCriterion($Rule$,$Theory$,$Examples$)
      Exit while
    EndIf
    $Examples = Examples \setminus Cover$
    $Theory = Theory \cup Rule$
EndWhile
$Theory$ = PostProcess($Theory$)
Output : $Theory$

2.4 Rule induction algorithms

This section describes some algorithms that are used to learn rule sets. We can find in the literature many rule induction methods. Here we can see two different families of mechanisms used to induce rule sets, describing for each of them a representative example of learning system.

2.4.1 Separate-and-conquer

This is probably the most common family of rule induction systems found in the literature. Basically, the methods following this idea apply an iterative process consisting in first generating a rule that covers a subset of the training examples and then removing all examples covered by the rule from the training set. This process is repeated iterativelly until there are no examples left to cover (although there are more sofisticated policies). The final rule set is the concatenation of the rules discovered at every iteration of the process. Fürnkranz (Fürnkranz, 1999) did a good review of separate-and-conquer systems, where a meta-algorithm of the separate-and-conquer methodology is proposed, shown in figure 2.7. Some examples of systems following this schema are the *AQ* family of learning systems (Michalski, 1969), *CN2* (Clark & Niblett, 1989) and *RIPPERk* (Cohen, 1995).

How is the classification process used in the resulting rule sets of these systems? In ancient

systems with binary classification (positive/negative examples) the system only induces the rules covering the positive examples. Therefore, the order of the rules is not relevant because all of them cover the same class. We can consider that exists another *virtual* rule covering all negative examples, which is a kind of **default class**.

If we have classification problems with more than two classes, the most common option is to use the rule set as a *decision list*, that is, an ordered rule set. The original *CN2* (Clark & Niblett, 1989) and the *AQ* family of learning systems (Michalski, 1969) use this approach. A later version of *CN2* (Clark & Boswell, 1991) can encode unordered rule sets, using the voting-style process based on the previous performance of the rules, described in subsubsection 2.3.1

Another option is to define a general order of classes, and apply the separate-and-conquer sequencial mechanism to each class (considering the examples belonging to the class as positive examples and all the other examples as negative). The *RIPPERk* system uses this approach, proposing a class ordering based on the proportion of examples of each class in the training set, starting with the least frequent class and ending with the most frequent one (which creates a single default rule).

To illustrate the reader with a complete example of a rule induction system, figure 2.8 shows the pseudocode of the ordered *CN2* version. In short, the algorithm that induces each rule maintains a pool of predicates (starting with the most general one, ie, totally irrelevant) and, iteratively, it tries to specialize each rule by adding a term of the kind $attribute_i = value_i^j$, until no better predicate can be found. Then, the class most covered by this predicate in the current set of examples is assigned to the predicate to construct a rule.

2.4.2 Learning all the rules at the same time

As an alternative to the separate-and-conquer strategy we can find sistems that evolve a whole set of rules in a single iterative process. An example of this strategy is the *RISE* system (Domingos, 1994). It presents itself as an hybrid between an instance-based and a rule-based system. The aim of the system is to model the concepts of the problem using rules, but maintainig a set of instances for the outlayer and special examples. The algorithm performs an iterative process of rule refining. The set of training examples is loaded as the initial set of rules, and the refining process consists in generalizing these rules-instances to cover other examples of the same class, removing subsumed rules in the process. The produced rules are unordered and the *Laplace accuracy* is used to choose the rule that performs the prediction for an input example. Figure 2.9 shows the algorithm of *RISE*.

Figure 2.8: The CN2 rule induction algorithm

Ordered CN2 learning algorithm
Input : $Examples, Classes$
$RuleSet = \emptyset$
Repeat
$BestPredicate =$ FindBestPredicate($Examples$)
  If $BestPredicate \neq \emptyset$
    $class =$ most covered class from $Classes$ in $Examples$
    $rule =$ Construct rule "**If** $BestPredicate$ **Then** predict $class$"
    $RuleSet = RuleSet \cup rule$
    $Examples = Examples \setminus$ Cover($BestPredicate$)
  EndIf
Until $BestPredicate = \emptyset$
Output : $RuleSet$

FindBestPredicate
Input : $Examples$
$mgc =$ Most general predicate ("true")
$star = \{mgc\}$
$BestPredicate = \emptyset$
While $star \neq \emptyset$
  $newStar = \emptyset$
  ForEach $pred$ in $star$ **Do**
    ForEach any attribute $test$ non existent in $pred$ **Do**
      $pred' =$ specialization of $pred$ adding $test$ to it
      If $pred'$ is better than $BestPredicate$ and $pred'$ is statistically significant
        $BestPredicate = pred'$
      EndIf
      $newStar = newStar \cup pred'$
      If Size($newStar$) $> maxStar$ (used defined)
        Remove worst condition from $newStar$
      EndIf
    EndForEach
  EndForEach
  $star = newStar$
EndWhile
Output : $BestPredicate$

Figure 2.9: The RISE rule induction algorithm

RISE learning algorithm
Input : $Examples$
$RuleSet = Examples$
Compute Accuracy($RuleSet$)
Repeat
  ForEach $rule$ in $RuleSet$ **Do**
    Find the nearest example ex to $rule$ not already covered by it and belonging to the same class
    $rule'$= MostSpecificGeneralization($rule$,ex)
    $RuleSet'$ = $RuleSet$ replacing $rule$ by $rule'$
    If Accuracy($RuleSet'$) > Accuracy($RuleSet$)
      $RuleSet = RuleSet'$
      If $rule'$ is identical to another rule in $RuleSet$
        Remove $rule'$ from $RuleSet$
      EndIf
    EndIf
  EndForEach
Until Accuracy($RuleSet$) cannot be increased
Output : $RuleSet$

MostSpecificGeneralization
Input : $Rule, Example$
// $Rule_1 \cdots Rule_n$ *are the tests assigned to each attribute.*
// *For nominal attributes* $Rule_i$ *is either* $true$ *(irrelevant) or* $Rule_i = value_i^j$.
// *For numeric attributes* $Rule_i = [Rule_{i,lower}, Rule_{i,upper}]$
ForEach attribute i in the domain **Do**
  If $Rule_i = true$
    do nothing
  Endif
  If attribute i is nominal and $Rule_i \neq Example_i$
    $Rule_i = true$
  EndIf
  If attribute i is numeric and $Example_i < Rule_{i,lower}$
    $Rule_{i,lower} = Example_i$
  EndIf
  If attribute i is numeric and $Example_i > Rule_{i,upper}$
    $Rule_{i,lower} = Example_i$
  EndIf
EndForEach
Output : $Rule$

2.5 Discretization algorithms

Sometimes, there are learning algorithms that are unable to handle real-valued attributes or that handle nominal attributes in a much easier way. If such a learning algorithm has to solve domains with real-valued attributes, a *discretization process* is needed. A discretization process transforms continuous-valued attributes into nominal ones by splitting the range of the attribute values in a finite number of intervals. The so found intervals are then used for treating continuous-valued attributes as nominal. Most discretization algorithms can be classified by the following criteria (Liu, Hussain, Tam, & Dash, 2002):

supervised/non-supervised A supervised discretization algorithm uses the class of the training examples to decide which cut-points it creates in the domain of the real-valued attributes. A non-supervised discretizer does not take into account the class.

dynamic/static In the context of supervised discretization, dynamic discretizers are the ones that perform the discretization task *while* the learing process is being done. On the other hand, a static discretization is applied *before* the learning process.

global/local A local discretization is applied only to a certain subset of the instance space, while a global one is applied to the full instance set to discretize.

splitting/merging The discretization process can be done in two different ways: starting with any possible cut-point in the domain of the attribute: the midde-points between all values of the attribute existing in the training set, and then merging some of these cut points under certain criteria, which is called *merging*. *Splitting* is the opposite method, it starts with a single (therefore irrelevant) attribute and it splits it under certain criteria. This process is repeated with the created intervals until some stop criterion.

A description of the discretization algorithms that are used in chapter 6 follows, although there are many more discretizers in the literature (Holte, 1993; Catlett, 1991; Ho & Scott, 1997; Chan, Batur, & Srinivasan, 1991; Liu & Setiono, 1995; Wang & Liu, 1998; Kozlov & Koller, 1997; Elomaa & Rousu, 2002; Yang & Webb, 2002).

Equal-width This is one of the simplest methods. The domain of the attribute is divided into n equal sized intervals, where n is a parameter. This is a non-supervised and splitting algorithm.

Equal-frequency As an alternative to the above discretizer, the n chosen intervals contain an equal number of values, in order to offer a better set of intervals where the value

distribution in the attribute domain is non-uniform. It is also non-supervised and of splitting class.

Id3 (Quinlan, 1986). This discretization algorithm takes its name from the decision tree learning system of the same name, because the criterion used to decide the cut points is the same as that used in the learning system to decide the attribute that will be used to partition the tree: the entropy minimization criteria:

$$Entropy(X) = -\sum_{x} P_x log_2(P_x) \tag{2.2}$$

$$P_x = \frac{|\{ins \in X | class(ins) = x\}|}{|X|} \tag{2.3}$$

The entropy metric (Shannon & Weaver, 1949) is applied to the training examples belonging to a certain partition of the attribute we are discretizing. The Id3 discretizer algorithm splits the attribute domain in a recursive way. The cut point chosen in each recursive call is the one that creates two partitions with minimum entropy as represented in equation 2.4. S is the interval being split and S_1 and S_2 are the partitions to the left and to the right of the tested cut point. The stop criteria is finding an interval where all contained examples belong to the same class. This is a splitting supervised algorithm.

$$EntropyPartition(S, S_1, S_2) = Entropy(S_1)\frac{|S_1|}{|S|} + Entropy(S_2)\frac{|S_2|}{|S|} \tag{2.4}$$

The Fayyad & Irani algorithm (Fayyad & Irani, 1993). This algorithm is an extension of *Id3*, changing the stop criterion to a most agressive one that usually generates significatively less number of cut-points, this is one of the most popular discretization algorithms in the literature. The new criterion is based on the *Minimum Description Length* (MDL) principle (Rissanen, 1978), a metric inspired in the information transmission field that balances the accuracy and complexity of a model in a sensible way. Recursive partitioning is stopped if the formula in equation 2.5 is true, where k_i is the number of classes defined in partition S_i.

$$Entropy(S) - EntropyPartition(S, S1, S2) < log_2\frac{(N-1)}{N} + \frac{\Delta(S, S_1, S_2)}{N} \tag{2.5}$$

$$\Delta(S, S_1, S_2) = log_2(3^k - 2) - [k \cdot Entropy(S) - k_1 \cdot Entropy(S_1) - k_2 \cdot Entropy(S_2)] \tag{2.6}$$

USD (Giráldez, Aguilar-Ruiz, Riquelme, Ferrer, & Rodríguez, 2002) divides the continuous attributes in a finite number of intervals with maximum goodness, so that the average-goodness of the final set of intervals will be the highest. The main process is divided in two different parts: first, it calculates the initial intervals by means of projections,

which will be refined later, depending on the goodnesses obtained after carrying out two possible actions: to join or not adjacent intervals. It is supervised and of a merging type.

Mantaras discretizer (Cerquides & de Mantaras, 1997). This method is analogous to the Fayyad & Irani one, but changes the metric used to decide a new partition to the *Mantaras distance* (De Mántaras, 1991), another entropy-based metric used previously, like *ID3* to induce decision trees (De Mántaras, 1991):

$$Dist(S_1, S_2) = 2 - \frac{Entropy(S_1) + Entropy(S_2)}{Entropy(S_1 \cap S_2)} \tag{2.7}$$

$$Entropy(S_1 \cap S_2) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij} log_2(P_{ij}) \tag{2.8}$$

$$P_{ij} = P_i \times P_j \tag{2.9}$$

ChiMerge (Kerber, 1992). This discretizer is based on the χ^2 statistical test, which performs a significance test on the relationship between the values of a feature and the class. The author argues that the class frequencies within an interval, measured by this statistic, should be different in adjacent intervals. If they are not, the intervals are merged. This is a supervised merging algorithm, starting with as many intervals as values, that are merged iteratively based on χ^2 until no more merges are possible. This method needs a parameter: the confidence level of the statistical test. The exact formulation of χ^2 is:

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{p} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{2.10}$$

$$E_{ij} = (R_i \times C_j)/N \tag{2.11}$$

$$R_i = \sum_{j=1}^{p} A_{ij} \tag{2.12}$$

$$C_j = \sum_{i=1}^{2} A_{ij} \tag{2.13}$$

$$N = \sum_{j=1}^{p} C_j \tag{2.14}$$

Where p is the number of classes and A_{ij} is the number of values in the interval i and class j.

2.6 Scaling-up of machine learning systems

When the volume of information in the training set starts to increase, the computational cost of the learning process can be enormous, depending on the theoretical cost of the algorithm. Given this situation, an adaptation of the current or new techniques is required in order to have a reasonable learning time. As usual, there are several ways to achieve this objective. Here we focus only on a subset of them: Those that use only a subset of the training examples to perform the learning process. Only these techniques are described because they are the closest ones to the contributions in run-time reduction proposed in this thesis:

- **Wrapper Methods** (Fürnkranz, 1998; Quinlan, 1993; John & Langley, 1996; Skalak, 1994; Sierra, Lazkano, Inza, Merino, Larrañaga, & Quiroga, 2001). These methods build a layer over the learning process which selects the correct subset of examples by running iteratively the unmodified learning algorithm. The subset of used training examples varies through the iterations until the stop criteria is achieved. This stop criteria usually is based on the estimation that the current subset of examples is similar enough to the whole set.

- **Modified learning algorithms** (Fürnkranz, 1998; Maloof & Michalski, 2000; Wilson & Martinez, 2000). In this category we include the methods that have been modified to include the incremental learning inside their algorithm or methods that include/discard training examples based on knowledge-representation specific information.

- **Prototype Selection** (John & Langley, 1996; Aguilar-Ruiz, Riquelme, & Toro, 2000; Salamó & Golobardes, 2002). The methods included in this category reduce the training set before the learning process. Thus, the learning is performed only once, unlike the two prior categories. It should be remarked that this definition is related to the structure of the incremental learning system, because the general definition of prototype selection is much broader than what we have state here. It is obvious that the performance of the learning system is strongly biased by the behavior of the prototype selection process.

2.6.1 Wrapper methods

A very common wrapper method is named *windowing algorithm* (Fürnkranz, 1998). This scheme defines an initial training set size (*window*) for the initial iteration and also a maximum increment size: the maximum number of examples that can be added to the previous set of examples at each iteration. At the end of each iteration the correctly classified examples are

removed from the window. The *C4.5* system (Quinlan, 1993) also follows this scheme for its windowed version.

The previously described systems stop iterating when the training examples not included in the window can be classified correctly, using the theory generated in the current iteration. The *dynamic sampling* method (John & Langley, 1996) by John and Langley uses a different approach. This method estimates the accuracy of the whole set and stops adding more examples to the window when the difference between the accuracy of the current subset and this estimation falls below a certain threshold. After each iteration, the K examples used to calculate the current accuracy (acting as a test set) are added to the training set.

The next approaches identify themselves as *prototype selection*, but they are included here instead of in the *prototype selection* category because the selection is achieved by iteratively running some learning algorithm, instead of using some other technique or heuristic.

First, two methods defined by Skalak (Skalak, 1994) as *Sampling* and *Random Mutation Hill Climbing*. Sampling uses the *Monte Carlo* statistical tool to select n samples of m examples from the training set and runs the learning algorithm with each sample. The sample which produces the theory with best accuracy is selected. The second method defines a binary string with one bit for each training example which selects/unselects it and uses a local search method, the *Random Mutation Hill Climbing* to find a good string. That is, a good subset of examples.

A similar method was defined by Sierra et al. (Sierra, Lazkano, Inza, Merino, Larrañaga, & Quiroga, 2001), but instead of *Random Mutation Hill Climbing* they use *Estimation of Distribution Algorithms*.

2.6.2 Modified learning algorithms

In this category we include the systems that either (a) have integrated the incremental part of the system inside the learning algorithm or (b) are wrapper methods which use knowledge representation specific information or the partial theory generated to add/discard examples from the training set for the next iteration.

We find with the name *Integrative Windowing*, developed by Fürnkranz (Fürnkranz, 1998), a wrapper method that differs from the general *Windowing* schema about how the final theory is generated. It is the union of the partials theories obtained at each iteration of the incremental learning. This approach is feasible because it uses separate-and-conquer learning algorithm and this makes easy to merge the partial theories. At the end of each iteration the training examples that are well covered by the accumulated theory are discarded, reducing the current window and the computational cost. The author also reports accuracy gain in the use of

Integrative Windowing and discusses the effect of noise in incremental learning and proposes a noise-tolerant version of his algorithm (*Noise Tolerant Windowing*).

A similar system was proposed by Maloof and Michalski (Maloof & Michalski, 2000) named *Partial Memory Learning*. This method defines some refined policies to include and forget examples from the current window (partial memory). These policies use information from the knowledge representation to detect the relevant and irrelevant examples.

We also include in this category several methods in the *Case-Based Reasoning* and *Instance-Based Learning* fields, usually identified as *Case Base Reduction* techniques. An extensive review of these techniques can be found in (Wilson & Martinez, 2000).

2.6.3 Prototype selection

This category includes the methods which reduce the training set before the learning algorithm is run. Thus, the learning algorithm does not need to be modified and it is only run once.

A very simple approach in this category is called *static sampling* (John & Langley, 1996). This method selects a sample of the training set and uses some statistical tests to determine if the sample is sufficiently similar to the whole training set. The χ^2 hypothesis test is used for categorical attributes and a large-sample test relying on the central limit theorem is used for numerical attributes.

Another approach which is suited for axis-parallel knowledge representations is *Editing by Ordered Projection* by Aguilar et al. (Aguilar-Ruiz, Riquelme, & Toro, 2000). It is an heuristic method based on a geometrical projection of the examples.

Finally, we include in this category the *Sort-Out Techniques* by Salamó and Golobardes (Salamó & Golobardes, 2002) which uses the data-mining method *Rough Sets* to reduce a priori the *Case Base* for its application to *Case-Based Reasoning*.

2.7 Handling missing values

Missing data is an important potential threat to learning and classification because it may compromise the ability of a system to develop robust, generalized models of the concept/s represented by the training set. Some of the more popular methods for handling missing data (Roth, 1994) appear below:

Value ignoring Consider as true any test involving a missing value

Listwise or casewise data deletion If a record has missing data for any one variable used in a particular analysis, omit that entire record from the analysis.

Mean substitution Substitute a variable mean value computed from available cases to fill in missing data values on the remaining cases. A more sophisticated version uses the variable mean of the instances belonging to the same class as the one with the missing value

Regression methods Develop a regression equation based on complete case data for a given variable, treating it as the outcome and using all other relevant variables as predictors. Then, for cases where Y is missing, plug the available data into the regression equation as predictors and substitute the predicted Y value into the database for use in other analyses.

Hot deck imputation Identify the most similar case to the case with a missing value and substitute Y value of this case for the missing case Y value.

Expectation Maximization (EM) An iterative procedure that proceeds in two discrete steps. First, in the expectation (E) step, the expected value of the complete example set likelihood is computed. In the maximization (M) step the expected values are substituted for the missing data obtained from the E step and then the likelihood fuction is maximized as if no data were missing to obtain new parameter estimates. The procedure iterates through these two steps until convergence is obtained.

Raw maximum likelihood Use all available data to generate maximum likelihood-based sufficient statistics. Usually these consist of a covariance matrix of the variables and a vector of means. This technique is also known as Full Information Maximum Likelihood (FIML).

2.8 Summary of the chapter

This chapter has described some base material of the area of the artifical intelligence field where this thesis is applied: machine learning (ML). The chapter started with a general description of the machine learning paradigms and with some definitions related to the learning task we are solving: classification problems. Later there was a description of some knowledge representations used in machine learning and the description of some kinds of rule-based learning algorithms. Finally three specific topics were described: Discretization algorithms, scaling-up of learning systems and handling of missing data.

The aim of this description has been to provide enough ML-wise background material to construct the contributions that will be presented in this thesis over it. This is the reason this description has not been a complete ML review, but instead it has been biased towards the techniques used in this thesis: rule-based inductive learning, discretization algorithms, scaling-up of learning algorithms and missing values.

The next chapter will have a similar structure, but will focus specifically on the machine learning paradigm used to perform our rule-induction tasks: evolutionary computation.

Chapter 3

Genetic Algorithms and Genetic-Based Machine Learning

The last chapter contained an overview of the artificial intelligence area is which this thesis is placed (machine learning) and the task which is the application of this thesis, classification problems. In this chapter we focus on the specific machine learning paradigm that is used in the contributions presented in this thesis: evolutionary learning. Evolutionary learning (also known as genetic-based machine learning (GBML)) is the application of evolutionary computation (EC) to learning tasks. Evolutionary computation is a field that gathers a large collection of techniques inspired in biological processes such as population-based evolution, natural selection and genetics. These techniques can be applied to several kind of tasks: search, optimization, scheduling, and, of course, machine learning.

The chapter is structured as follows: section 3.1 will contain a brief description in general of the EC field and also a description of the main mechanisms of the most popular EC paradigm: the genetic algorithms (GA). Next, section 3.2 will show a bit of GA *theory and a formal methodology of the application of* GA. *The rest of the chapter will be focused specifically on GBML-related contents. The machine learning contents will start by describing three models of learning system in section 3.3. After the description of the models the thesis will be focused on some specific issues that are very related to the contributions presented in this thesis: representations for real-valued attributes in section 3.4, the scaling up of GBML systems in section 3.5 and the handling of the bloat effect in section 3.6. Finally, section 3.7 will provide a summary of the chapter.*

3.1 Introduction to Evolutionary computation and genetic algorithms

Evolutionary computation (EC) techniques are optimization tools that solve problems using procedures inspired by natural processes. These techniques usually work by transforming a population of individuals, being each individual a candidate solution for our problem. This transformation process consists in the iterative application of a cycle of stages inspired in natural selection and also in the generation of new individuals by genetic recombination. This combination of selection and recombination produces a directed exploration of the search space, converging to the regions of the space where the best solutions are placed. Research in evolutionary computation started in the 1960's, and the first major milestone was John Holland's book, considered a foundation work in the field (Holland, 1975).

3.1.1 Natural principles

Nature has been always able to solve one kind of task: survival and adaptation to the environment. Since life appeared on earth, the existing species have evolved, adapting themselves to where they live and becoming robust to changes. That is, the species were able to solve the problem of environment adaptation. In order to understand how the adaptation process of nature works, the work of Darwin and Mendel must be considered.

Charles Darwin proposed the concept of *natural selection*: the strongest individuals of a population (the better adapted to the environment) are the ones that survive. This process by itself has one problem: If the strong individuals dominate the weak completely, they will take over the population until all individuals are equal, which stops the adaptation process. Therefore the concept of diversity is necessary.

Mendel discovered that parents transmit their biological information to the offspring in the reproduction process. All this information necessary to define an individual is codified at cellular level in a structure called *chromosome*. Parts of the chromosome codify hair color, height, etc. New individuals are created by mixing the genetic information of the parents in a process called *crossover*. Therefore, new individuals are a mix of the information of the parents. However, this does not account for the problem stated above, that is, the crossover of two identical individuals means producing two identical offspring. A mechanism that introduces diversity is necessary: mutation.

Mutation can be defined as some small mistakes introduced during the mixing process of crossover. Thanks to mutation, new information that did not exist in the parents is introduced.

Sometimes this change creates worse individuals, but sometimes it creates better ones, who are the next step in the evolutionary process.

How are all these concepts are related to artificial intelligence? They are the source of inspiration for the processes involved in the evolutionary computation techniques.

3.1.2 Evolutionary computation and its paradigms

These natural principles mentioned above have inspired the techniques gathered with the name Evolutionary Computation. These techniques share some concepts with their biological inspiration, but also have some important differences. Using a classical classification, we can describe four main EC paradigms (Freitas, 2002):

Evolution Strategies (ES) (Rechenberg, 1973). These techniques typically use an individual representation consisting of a real-valued vector. Early ES emphasized mutation as the main exploratory search operator, but currently both mutation and crossover are used. An individual often represents not only real-valued variables of the problem being solved but also parameters controlling the mutation distribution, characterizing a self-adaptation of mutation parameters. The mutation operator usually modifies individuals according to a multivariate normal distribution, where small mutations are more likely than large mutations.

Evolutionary Programming (EP) (Fogel, 1964). Originally developed to evolve finite-state machines, but it is now often used to evolve individuals consisting of a real-valued vector. Unlike ES, in general it does not use crossover. Similar to ES, it also uses normally-distributed mutations and self-adaptation of mutation parameters.

Genetic Algorithms (GA) (Holland, 1975; Goldberg, 1989a). This is the most popular paradigm of EC. GAs emphasize crossover as the main exploratory search operator and consider mutation as a minor operator, typically applied with a very low probability. In early ("classic") GAs individuals were represented by binary strings, but nowadays more elaborate representations, such as real-valued strings, are also used.

Genetic Programming (GP) (Koza, 1992). This paradigm is often described as a variation of GAs rather than a mainstream EC paradigm in an of itself. Individuals being evolved in this paradigm are various kinds of computer programs, consisting not only of data structures but also of functions (or operations) applied to those data structures. These programs are usually represented using trees.

Figure 3.1: Main code of a simple genetic algorithm

```
Genetic Algorithm
t:=0
initialize P(t)
evaluate P(t)
Whileend Condition(P(t)) is not true
        t:=t+1
        P'(t)=Select a parent population from P(t)
        Apply crossover to P'(t)
        Apply mutation to P'(t)
        Evaluate P'(t)
        P(t+1)=Replacement(P(t),P'(t))
EndWhile
Output : best individual of P(t)
```

In recent years a new paradigm has been developed, which could be added to the previous list. It is known as *estimation of distribution algorithms (EDAs)* (Larranaga & Lozano, 2002). The main difference from the above stated paradigms are the recombination operators used: An statistical model is created from the individuals of the population, and the offspring are generated by sampling this model. Thus, the exploration process is less *blind* than the one used in the other EC paradigms.

As genetic algorithms are the focus of this thesis, the next subsection will describe GAs basic mechanisms.

3.1.3 Description of the basic mechanisms of GAs

Algorithmically, we can define GA (Goldberg, 1989a) as represented in figure 3.1.

The concepts that define a genetic algorithm are:

Individual A candidate solution to the problem we are solving

Chromosome The codification of an individual. Usually individuals, unlike in nature, are codified using a single chromosome

Gene Each of the atomic values of a chromosome

Fitness function The function that indicates the degree of adaptation of an individual to the environment where it lives. That is, how good is the individual in solving the problem.

Parent selection The process that chooses the most fitted individuals to the environment to produce offspring. This process uses the value given by the fitness function to each individual to decide which are the most fit individuals. There are many selection algorithms, some of them choose individuals based on their proportion of fitness value over the whole population, other methods are rank based, and only take into account if an individual is better than another, not how much better it is.

Crossover A process inspired in natural reproduction. Parents mix their chromosomes to create the offspring. Usually there is some probability (p_c) of a candidate parent producing offspring. The mix can be performed in several ways. The most classical one, the one-point crossover, chooses randomly a cut point in the crossover and creates offspring by mixing the contents to the left of this point from one parent with the contents to the right of the point from the other parent

Mutation The alteration of the genetic material of an individual. Like crossover, this operator is controlled by a certain probability (p_m), usually gene-wise. For binary representations, the most typical mutation is to flip a gene, changing from 1 to 0 or from 0 to 1.

Replacement The process that given an original population and an offspring's population, merges them to create the population for next iterations. The most classical approach is to use only the offspring population, and the best individual of the original population

3.2 Basic theory of GA and a formal methodology for its use

In the literature there are many examples of addressing the development of a formal theory for the behaviour and convergence of GAs (Rudolph, 1998; Vose, 1999). One of the oldest but also well accepted theory was proposed by Holland (Holland, 1975), and it is called *the schema theorem*.

The theorem is based on the concept of schema, a meta-representation of a chromosome. It is a string, of the same length of the chromosome, build using a ternary alphabet $0, 1, *$. Values 0 and 1 represent specified values in the chromosome. $*$ represents a "don't care", a position that can take either 0 or 1. For example, the schema $00*10$ is represented by two chromosomes: 00010 and 00110.

In order to state the theorem some definitions are needed:

Order $o(h)$ of a schema h is the number of specified positions, that is, not containing an asterisk.

Defining length $\delta(h)$ of a schema h is the distance between the outermost non-asterisks symbols. A schema with a high defining length has more risk of being broken, because it has more chances of having a cut-point between the specified positions.

The schema theorem describes how the frequency of schema instances changes with the iterations considering the effects of selection, crossover and mutation, supposing fitness-proportionate selection, one-point crossover and gene-wise mutation probability. The schema theorem is defined as:

$$E(m(h,t+1)) \geq m(h,t) \cdot \frac{f(h)}{\bar{f}} \cdot \left[1 - P_c \cdot \frac{\delta(h)}{l-1}\right] \cdot [1 - P_m]^{o(h)} \tag{3.1}$$

Where $m(h,t)$ is the number of instances of schema h in the population at time t, $f(h)$ is the average fitness of the instances of schema h and $\bar{f}$ is the average fitness of the population and l is the length of the chromosome. The effect of the fitness proportionate selection is expressed by $f(h)/\bar{f}$, indicating that over-the-average schemata should increase its number of instances in next population. The effect of crossover is represented by $\left[1 - P_c \cdot \frac{\delta(h)}{l-1}\right]$, indicating the chances of survival of the schema depending on the probability of crossover and its defining length. Finally, the effect of mutation is defined by $[1 - P_m]^{o(h)}$, indicating the probability of mutation not flipping any of the specified positions (the order of a schema).

The schema theorem states that the frequency of schemata with fitness higher than the average, short defining length and low order increases in the next generation. Departing from this theorem, Goldberg proposed the *building block hypothesis* (Goldberg, 1989a), that states that "short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations", defining these low-order subsets of a schemata as *building blocks*.

Moreover, Goldberg states the success of a GA using a methodology of seven problems that need to be handled (Goldberg, 2002), all of them dealing with building blocks (BB):

1. Know what GAs process – building blocks
2. Know thy BB challengers – building-block-wise difficult problems
3. Ensure an adequate supply of raw BBs
4. ensure increased market share for superior BBs
5. Know BB takeover and convergence times
6. Make decisions well among competing BBs

7. Mix BB well

Answering these questions leads to a facet-wise analysis of the GA, and the proposal of several models that all together have the objective of guaranteeing the success of a GA for problems of bounded difficulty. Some of these models are for population sizing (Harik, Cantu-Paz, Goldberg, & Miller, 1997; Goldberg, Sastry, & Latoza, 2001) or convergence time (Bäck, 1995). However, the last problem is difficult to address with a simple genetic algorithm for complex problems (Goldberg, 2002), because standard crossover operator cannot handle one task correctly: linkage learning, which is the process of identifying which genes are coupled among them. In short: Building Block identification. New recombination methods that could be able to discover what genes belong to each BBs are needed. Some examples of these methods are the *Linkage Learning Genetic Algorithm* (Harik, 1995) and the *Bayesian Optimization Algorithm* (Pelikan, Goldberg, & Cantú-Paz, 1999).

3.3 Three models of GBML learning systems

The rest of the chapter is focused specifically on issues related to machine learning. The first step, in this section, is to describe some general models or approaches to apply evolutionary computation techniques to machine learning tasks. As stated previously in the introduction chapter of this thesis, usually only two models are described in the literature, called *Michigan approach* and *Pittsburgh approach*. However, in recent years a third approach, called *Iterative Rule Learning*, has risen in popularity. This third approach, first used in the *SIA* system (Venturini, 1993) uses the separate-and-conquer methodology (explained in chapter 2) quite popular in the non-evolutionary rule induction field. The next subsections will detail each of these GBML models.

In this approach an individual is a rule, like in *Michigan*, but the solution provided by the *GA* is the best individual of the population, like in *Pitt*, although the final solution is the concatenation of the rules obtained by running the *GA* several time.

3.3.1 The Pitt approach

The Pitt approach proposes a GBML system using the traditional cycle of a *GA*, where each individual is a complete solution to the classification problem. A complete solution to the classification problem means a disjunction of a set of classification rules. Each rule has fixed length, but the number of rules of the set is variable. Individuals compete among themselves to correctly classify the maximum amount of training examples and the population converges towards good rule sets. Two representative systems of this family are:

- GABIL (DeJong & Spears, 1991)
- GIL (Janikow, 1991)

GABIL

- Knowledge representation
  - Each individual is a variable-length set of rules:

$$I = (R_1 \vee R_2 \ldots \vee R_n)$$

  - Each classification rule has binary representation, fixed length and codifies a predicate. This system performs concept learning from positive/negative examples. Rules only cover the positive examples, thus, there is no class associated to the rule. The semantic representation of the rule is:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee A_n = V_m^b))$$

  A_i, $i \in [1..n]$ is the attribute i of the dataset
  A_i^j,$i \in [1..n]$ i $j \in [1..m]$ is the value j that can take the attribute i

  - These predicates can be mapped to a binary string with the following procedure:
    * We have 4 attributes: (A1,A2,A3,A4). The values of A1 are (A,B,C,D), the values of A2 are (E,F,G), the values of A3 and (H,I,J,K,L) and finally the values of A4 are (M,N).
    * The predicate "(A1 is B or C) and (A2 is E or F or G) and (A3 is H i K) and (A4 is M)" is represented as:

$A1$	$A2$	$A3$	$A4$
0110	111	10010	10

  - Looking at the examples we can see that all bits associated to the attribute $A2$ are set to 1. This is the mechanism that the representation has to indicate that this attribute is irrelevant.

- Fitness function. The fitness function is computed after classifying all instances of the training set, and consists simply of a squared accuracy function:

$$fitness(individual) = \left(\frac{\#\text{instances correctly classified}}{\text{total } \#\text{instances}}\right)^2$$

- Crossover operator. This operator needs a small restriction to guarantee that semantically-correct offspring are created. Cut points can take place in any rule of the individual, which does not have to be the same for both parents, but it has to be placed in the same position *inside the rule*.

- Variants of the system

  - **GABL** (GA Batch concept learner). It is the system as described so far.
  - **GABIL** (GA Batch-incremental concept learner). It is an evolution of *GABL* with an incremental learning process:
    * The system starts learning with only one training example. A rule set is generated covering it.
    * After generating the initial rule set, the system tries to classify a second example with it.
    * If the new examples is classified correctly, the same test is repeated with more examples.
    * If not, *GABL* is run again, using all the instances tested so far.

GIL

- Knowledge representation
  This system also evolves a disjunction of rules. Each rule contains a predicate defined in the VL_1 logic (Michalski, Mozetic, Hong, & Lavrac, 1986), although the mapping into a binary string is equivalent to the one used in *GABIL*

- Fitness function.
  The fitness function used in *GIL* has as a goal balancing the accuracy and complexity of the individuals by means of the product of two terms related to these measures:

$$fitness = correctness \cdot (1 + w_3 \cdot (1 - cost))^f \tag{3.2}$$

f grows very slowly on $[0, 1]$ as the population ages, and correctness is defined as:

$$correctness = \frac{w_1 \cdot \epsilon^+/E^+ + w_2 \cdot (1 - \epsilon^-/E^-)}{w_1 + w_2} \tag{3.3}$$

Where ϵ^+/ϵ^- is the number of positive/negative examples covered by the individual and E^+/E^+ is the number of positive/negative examples in the dataset. Cost is defined as:

$$cost = 2 \cdot \#rules + \#conditions \tag{3.4}$$

The reported value (Janikow, 1993) for both w_1 and w_2 is 0.5. w_3 takes smaller values, such as 0.01-0.02.

- Genetic operators. This system is very rich in high-level operators that modify the chromosome at the semantic level, unlike the "traditional GA operators" used in *GABIL*. The operators can be classified in 3 categories:
  - Chromosome level
    - RuleExchange: this operator exchange complete rules between two parents. For example, the two parents:
      $< 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11 >$
      $< 111|001|01|111|1111|01 \vee 110|100|10|111|0010|01 >$
      can generate the two following sons:
      $< 100|111|11|111|1000|11 \vee 111|001|01|111|1111|01 >$
      $< 010|111|11|010|1111|11 \vee 110|100|10|111|0010|01 >$
    - RuleCopy: this operator removes a rule from a parent and appends it to the other one:
      $< 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11 >$
      $< 111|001|01|111|1111|01 \vee 110|100|10|111|0010|01 >$
      can generate the two following sons:
      $< 100|111|11|111|1000|11 \vee 111|001|01|111|1111|01 \vee 110|100|10|111|0010|01 >$
      $< 010|111|11|010|1111|11 >$
    - NewPEvent: unary operator that given an individual and an uncovered positive example, appends it to the rule set:
      Individual $< 100|111|11|111|1000|11 \vee 111|001|01|111|1111|01 >$
      and instance :

$< 100|010|10|010|0010|01 >$
produce:
$< 100|111|11|111|1000|11 \vee 111|001|01|111|1111|01 \vee 100|010|10|010|0010|01 >$

* RuleGeneralization: unary operator that generalizes a random subset of the rules of an individual:
$< 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11 \vee 100|010|10|010|0010|01 >$
Choosing rules 2 and 3, the system produces::
$< 100|111|11|111|1000|11 \vee 110|111|11|010|1111|11 >$

* RuleDrop: unary operator that eliminates a random subset of the rules of an individual:
$< 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11 \vee 100|010|10|010|0010|01 >$
may produce :
$< 100|111|11|111|1000|11 >$

* RuleSpecialization: unary operator that specializes a random subset of the rules of an individual:
$< 100|111|11|111|1000|11 \vee 010|111|11|010|1111|10 \vee 111|010|10|010|1111|11 >$
Choosing rules 2,3 it may produce :
$< 100|111|11|111|1000|11 \vee 010|010|10|010|1111|10 >$

– At rule level:

* RuleSplit: this operator splits a rule in two: $< 100|111|11|111|1000|11 >$
Dividing the rule by the second attribute:
$< 100|011|11|111|1000|11 \vee < 100|100|11|111|1000|11 >$

* SelectorDrop: this operator selects an attribute and makes it irrelevant (all bits set to 1):
$< 100|111|11|111|1000|11 >$
Choosing attribute 5:
$< 100|011|11|111|1111|11 >$

* IntroSelector: this operator selects an irrelevant attribute to specialize it:
$< 100|111|11|111|1111|11 >$

Choosing attribute 5:
$< 100|011|11|111|0001|11 >$

* NewNEvent: this operator, given a negative example covered by the rule, splits the rule to uncover it:
Given the rule:
$< 110|010|11|111|1111|11 >$
And the negative example:
$< 100|010|10|010|0100|10 >$
It produces the following rules:
$< 010|010|11|111|1111|11 \vee 110|0100|01|111|1111|11$
$\vee 110|010|11|101|1111|11 \vee 110|010|11|111|1011|11$
$\vee 110|010|11|111|1111|01 >$

– At attribute level

* ReferenceChange: operator that changes the state (0 or 1) of one of the values of an attribute:
$< 100|010|11|111|0001|11 >$
Changing the third value of the forth attribute:
$< 100|010|11|110|0001|11 >$

* ReferenceExtension: operator that adds ones (generalizes) to an attribute:
$< 100|010|11|111|1010|11 >$
Modifying attribute 5:
$< 100|010|11|110|1110|11 >$

* ReferenceRestriction: operator that removes ones (specialize) from an attribute:
$< 100|010|11|111|1011|11 >$
Modifying attribute 5:
$< 100|010|11|110|1000|11 >$

Most of these operators have associated parameters (probabilities), which makes the GIL system, in comparison with GABIL, more complex to tune.

3.3.2 The Michigan approach

The Michigan approach is characterized by the creation of a cognitive model called *classifier system* where the population members are individual rules, and the whole population is the solution to the classification problem. This means that a mechanism that rewards good rules and penalizes bad rules is needed, usually based on reinforcement learning techniques. In this approach the GA is not the central element, but only a part of the system used from time to time to discover new rules.

Nowadays the most popular Michigan system is called *XCS* (Wilson, 1995). It is an evolution of *ZCS* by the same author. The fitness of the individuals is based on the prediction accuracy of each rule. A full description of the system follows:

- Populations of rules used in the system:
  There are three kinds of populations used in different stages of the system:
  - $[P]$: is the population that contains all the rules evolved. This population can be generated randomly, from a set of known rules, empty or having, for each class, a single totally general rule.
  - $[M]$: this is the *match set*, the set of rules activated by an input instance.
  - $[A]$: once we have selected a predicted class from the rules in $[M]$, the *action set* $[A]$ is created containing all the rules that predict this class.
  - $[A]_{-1}$ is the *action set* of the previous cycle.
- Rule representation:
  Each rule (*classifier* in the XCS nomenclature) has a condition part and an action part. In classification problems the action part is an associated class. The condition is a string with as many symbols as attributes that, for binary problems, is built using the ternary alphabet $\{\text{“}1\text{”}, \text{“}0\text{”}, \text{“}\#\text{”}\}$. The $\#$ symbol indicates that the value of the associated attribute is irrelevant. Each classifier has also some associated parameters:
  - *p*: expected reward of the classifier if it classifies correctly an example
  - *e*: estimation of the prediction error of the classifier
  - *f*: fitness of a classifier
  - *exp*: number of times this classifier has been in an *action set* since it was created.
  - *ts*: Last time the GA was used on an action set where this classifier was involved.
  - *as*: estimated size of the action sets where this classifier has participated.

  - *num*: number of "micro-classifiers" that this "macro-classifier" represents. Rules in *XCS* can assimilate other classifiers that cover a subset of its examples, creating "macro-classifiers".

- Parameters of XCS
  - *N*: is the maximum size of the population
  - β is the learning rate for *p*, *e* and *f*
  - α_1, e_0 and v are used to compute the fitness of the classifiers
  - θ_{GA} is the activation threshold of the GA. GA is activated if the average time since it was last used is higher than this threshold
  - θ_{del} is the classifier deletion threshold. If the *exp* value of a classifier is higher that this threshold, it can be eliminated depending on its fitness.
  - δ is the percentage under the fitness average of $[P]$ where the fitness of a classifier modifies its deletion probability.

- Working cycle

1. We build $[M]$ from $[P]$ with each input example
2. If $[M]$ is empty or some of the classes are not predicted in $[M]$, the *covering* process is activated, and a new classifier is created, having as a condition a generalized version of the input example and using a class not covered in $[M]$. This classifier is introduced in the population
3. From $[M]$, a prediction for each class is made, based on the p and f values of each classifier.
4. A class is chosen from the calculated predictions, and $[A]$ is build. The system returns a payoff based on the prediction. The parameters of classifiers in $[A]$ is updated in the following way:

$$p \leftarrow p + \beta(R - p) \tag{3.5}$$

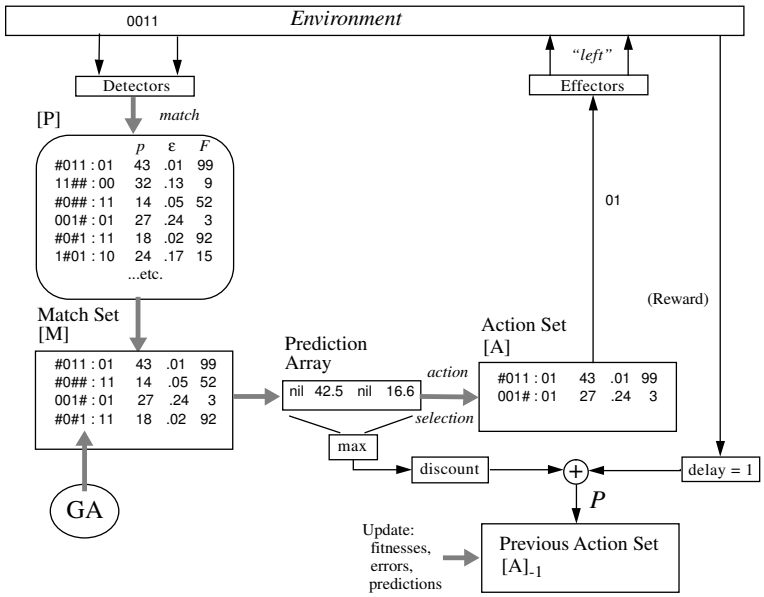
$$e \leftarrow e + \beta(|R - p| - e) \tag{3.6}$$

$$k = \begin{cases} 1 & \text{if } e < e_0 \\ \alpha(e_0/e)^v & otherwise \end{cases} \tag{3.7}$$

$$k' = \frac{k}{\sum_{x \in [A]} k_x} \tag{3.8}$$

$$F \leftarrow F + \beta(k' - F) \tag{3.9}$$

Figure 3.2: XCS working cycle



XCS cycle is represented in figure 3.2

- The genetic algorithm
  The GA is activated periodically based on θ_{GA}, and it is applied over $[A]$. It selects two parents from $[A]$ and performs a complete GA cycle, but the two offspring are inserted in $[P]$ without replacing their parents. If the number of classifiers in $[P]$ reaches N, some classifies must be deleted

- Choosing the classifiers to be deleted.
  Each classifier has a deletion probability, computed as follows:

  1. The deletion probability of a classifier is computed to be proportional to the estimation of the size of the match sets where the classifier can appear. In this way, all classifier subpopulations are given equal number of resources
  2. The probability can be increased if the classifier is experienced enough and its fitness is much lower than the average fitness of $[P]$.

- Macro-classifiers.
  When introducing new individuals in the population, the system checks if the new individual is equal to another one. In this case, the new classifier is not introduced, and the numerosity of the equal classifier is increased.

3.3.3 Iterative Rule Learning approach

The *Iterative Rule Learning*, first used in the *SIA* system (Venturini, 1993) uses the separate-and-conquer methodology (explained in chapter 2) to induce rules, using a *GA* to generate each rule. In this approach an individual is a rule, like in *Michigan*, but the solution provided by the *GA* is the best individual of the population, like in *Pitt*, although the final solution is the concatenation of the rules obtained by running the *GA* several time. This approach has been use extensively in genetic-fuzzy systems (Cordón, Herrera, Hoffmann, & Magdalena, 2001) but there are also some examples of application of this model to crisp representations, like the *HIDER* system (Aguilar-Ruiz, Riquelme, & Toro, 2003). A description of this system follows:

- Representation:
  This system uses intervals defined as $[lower, upper]$ for real-valued attributes, and a binary representation like the one in *GABIL* and *GIL* for nominal ones. Rules have an associated class, unlike these two systems.

- Initialization:
  Each rule is initialized picking randomly a training example, and assuring that the intervals/disjunctions of nominal values of the rule cover it.

- General structure:
  HIDER has two general stages. In the main level there is a separate-and-conquer style algorithm. In the inner level there is a GA inducing each rule. The code for both stages is represented in figure 3.3. The parameter epf ensures that no rule is created if the number of examples still to cover is too low, to avoid creating over-specific rules. The best individual of the parent population is preserved.

- Crossover operator:
  The crossover operator for real-valued attributes is an extension of Radcliffe's flat crossover (Radcliffe, 1990) that guarantees the semantic correctness of the intervals in the rule. For nominal attributes uniform crossover (Syswerda, 1989) is used.

- Mutation:
  If mutation affects a real-valued gene, some offset determined by a distance metric is added or subtracted from the gene value. For nominal genes, the value is changed from 0 to 1 or 1 to 0.

- Fitness function:
  The goal of the fitness function is two-fold: maximizing the number of covered examples

and minimizing the number of classification mistakes over the training set of the rule:

$$f(\varphi) = 2(N - CE(\varphi)) + G(\varphi) + coverage(\varphi) \tag{3.10}$$

Where φ is an individual, N is the total number of training examples, $CE(\varphi)$ is the number of wrongly classified examples and $G(\varphi)$ the number of correctly classifier examples. $coverage(\varphi)$ is computed as follows:

$$coverage(\varphi) = \prod_{i=1}^{m} \frac{coverage(\varphi, i)}{range(\varphi, i)} \tag{3.11}$$

$$coverage(\varphi, i) = \begin{cases} upper_i - lower_i & \text{attribute i is continuous} \\ k_i & \text{attribute i is nominal} \end{cases} \tag{3.12}$$

$$range(\varphi, i) = \begin{cases} U_i - L_i & \text{attribute i is continuous} \\ |A_i| & \text{attribute i is nominal} \end{cases} \tag{3.13}$$

$$\tag{3.14}$$

Where k_i is the number of active values of attribute i of φ, U_i is the upper bound of the domain of the attribute i, L_i is the lower bound of the domain of the attribute i and $|A_i|$ is the number of values of nominal attribute i.

3.4 Representations for real-valued attributes

Apart from the last example, all described systems use only discrete representations. In recent years, several representations for real-valued attributes have been proposed. In this section some examples of these representations will be described. Without the aim of excluding other systems, we can classify them in four categories:

Rules with real-valued intervals (Corcoran & Sen, 1994; Wilson, 1999; Stone & Bull, 2003; Aguilar-Ruiz, Riquelme, & Toro, 2003; Giráldez, Aguilar-Ruiz, & Riquelme, 2003; Divina, Keijzer, & Marchiori, 2003)

Decision trees with relational decision nodes (Llorà & Garrell, 2001b; Llorá & Wilson, 2004; Cantu-Paz & Kamath, 2003)

Synthetic sets of prototypes (Llorà & Garrell, 2001a)

Fuzzy representations (Cordón, Herrera, Hoffmann, & Magdalena, 2001)

Figure 3.3: The HIDER iterative rule learning algorithm

HIDER
Input : $Examples$
$RuleSet = \emptyset$
$n = |Examples|$
While $|Examples| = n \times epf$
    $rule =$ EvoAlg($Examples$)
    $RuleSet = RuleSet \cup rule$
    $Examples = Examples \setminus$ Cover($rule$)
EndWhile
Output : $RuleSet$

EvoAlg
Input : $Examples$
$i = 0$
$P_0 =$ Initialize()
Evaluation(P_0,i)
While $i < num - generations$
    $i = i + 1$
    For $j \in 1, \ldots, |P_{i-1}|$
        $\bar{x} =$ Selection(P_{i-1}, i, j)
        $P_i =$ Recombination($\bar{x}, P_{i-1}, i, j$)
    EndFor
Evaluation(P_i,i)
EndWhile
Output : $BestOf(P_i)$

3.4.1 Rules with real-valued intervals

From a semantic point of view we can consider that all systems in this category evolve rules that have the following structure:

$$\text{If } A_1 \in [l_1, u_1] \wedge A_2 \in [l_2, u_2] \wedge \cdots \wedge A_n \in [l_n, u_n] \text{ Then predict class } c_m \tag{3.15}$$

Where A_i is an attribute of the domain and l_i, u_i are the lower and upper bounds of an interval associated to the attribute i.

Can l_i and u_i take any value in the attribute domain? The answer to this question creates again two sub-categories:

- Representations based on discretization
- Representations handling directly real-valued bounds

Representations based on discretization

The aim of these representations is to reduce the search space of the problem by considering the set of cut points provided by a discretization algorithm as the possible bounds of the evolved intervals. In a certain way, it can be considered that they are performing a dynamic discretization.

A first example (Giráldez, Aguilar-Ruiz, & Riquelme, 2003) is an extension of the *HIDER* system described in the previous section, called *HIDER**. The authors use their own discretization algorithm, called *USD* (Giráldez, Aguilar-Ruiz, Riquelme, Ferrer, & Rodríguez, 2002), to create the candidate interval bounds. The most interesting feature is the codification used, which is called *natural coding*: Each possible interval created by these cut points is given a number. Rules do not contain the intervals themselves, but these numbers. Transition tables are created for crossover and mutation to determine how these numbers are changed. In practical usage, these transition tables add or subtract cut points from the interval codified by the number (for mutation) or create intervals that are the intersection of the parents (for crossover).

Another example is the *ECL* system (Divina, Keijzer, & Marchiori, 2003). This system is a hybrid evolutionary algorithm for rule induction. Its cycle of application is formed by selection, mutation and optimization. The mutation operators applied do not act randomly, but consider a number of mutation possibilities, and apply the one yielding the best improvement in the fitness of the individual. The optimization phase consists in a repeated application of mutation operators until the fitness of the individual does not worsen, or until a maximum number of

optimization steps has been reached. In the former case the last mutation applied is retracted. Numerical values are handled by means of inequalities, which describes discretization intervals. Inequalities can be initialized to a given discretization interval, e.g., found with the application of the Fayyad & Irani's algorithm (Fayyad & Irani, 1993). Inequalities are modified by the mutation operator in a similar way to the natural coding described above.

Representations handling directly real values

The main differences among the representations that evolve real-valued intervals are the ways in which the bounds of the interval are codified. The previous section already explained the representations used in *HIDER* (Aguilar-Ruiz, Riquelme, & Toro, 2003), using two genes to codify the lower and the upper bound, and a special crossover operator to guarantee the semantic correctness of the interval ($lower < upper$). Another approach (Corcoran & Sen, 1994) uses a standard crossover operator and considers the intervals where the upper bound is smaller than the lower one as irrelevant. A third way (Stone & Bull, 2003) to codify an interval with $upper$ and $lower$ bounds is to use non-fixed positions for the bounds. The lower value of the two genes associated to an interval becomes the lower bound, and the higher value the upper one. A different approach that does not need any kind of mechanism to guarantee the semantic consistency is used in the *XCSR* system (Wilson, 1999). In this system the interval is codified as a pair of real values defines as $center$ and $spread$. The lower bound of the interval is defined as $center - spread$ and the higher bound as $center + spread$.

3.4.2 Decision trees with relational decision nodes

There are different ways to apply GAs to induce decision-trees with real-valued tests. The *GALE* system (Llorà & Garrell, 2001b) is a fine-grained parallel genetic algorithm, with several knowledge representations that can co-evolve in a 2D board. One of these representations evolves full decision trees by means of genetic programming operators. The tests used in the internal nodes of the tree can be of three ways:

axis-parallel ($a_i \leq \theta$) where a_i is an attribute and θ a threshold

oblique ($\sum_{i_1}^{d} w_i a_i + w_{d+1} > 0$) where $\{w_1, w_2, \ldots, w_{d+1}\}$ is a vector of coefficients defining an oblique hyperplane

Originally, only one kind of test was used for all nodes of a tree, but in recent work (Llorá & Wilson, 2004) the mix of axis-parallel and oblique tests in the same decision tree has been studied.

A totally different approach (Cantu-Paz & Kamath, 2003), where the GA is not the central piece, is the heuristic construction of the structure of the tree followed by a an optimization by GAs and ES of the tests performed at each node.

3.4.3 Synthetic sets of prototypes

Another of the knowledge representations used in the *GALE* system (Llorà & Garrell, 2001b) consists of evolving a set of synthetic instances, and using a *k-nearest-neighbour* classifier to classify input instances. These prototypes do not need to have all attributes completely defined, and a partially-defined distance function is provided:

$$distance(ins, prot) = \sqrt{\frac{1}{|\pi(prot)|} \sum_{a \in \pi(prot)} \left(\frac{ins_a - prot_a}{domain_a} \right)^2} \tag{3.16}$$

Where $\pi(prot)$ is a set containing the defined attributes of $prot$ and $domain_a$ is the domain of the attribute a for the dataset. This distance function is basically a Euclidean distance adapted to partially-defined prototypes.

3.4.4 Fuzzy representations

In the literature there is extensive work on the integration of fuzzy logic (Zadeh, 1965) with evolutionary computation techniques, for classification and regression tasks. Good and general reviews of these techniques exist in the literature (Cordón, Herrera, Hoffmann, & Magdalena, 2001). A fuzzy system consists of two parts: the rule base (a set of fuzzy rules) and the data base (a set of linguistic variables used in the rules, each of them with an associated membership function). Evolutionary computation techniques can be used for:

- Evolving the rule base
- Evolving the data base
- Evolving both the fuzzy base and the data base

Also, we can find systems using the three learning models described in this chapter, Pittsburgh, Michigan and Iterative Rule learning.

3.5 The scaling-up of GBML systems

This section is equivalent to the section of the same name in previous chapter, but focused on GBML. The *GABIL* (DeJong, Spears, & Gordon, 1993) system described in section 3.3 is a good example of a *wrapper method*. However, its working mechanism of resetting the system for each wrongly classified example makes it not suitable for most modern-day real datasets, where perfect accuracy is hardly ever achieved. Therefore, most modern systems concentrate on the approaches that we called *modified learning algorithms* and *prototype selection*. Alex Freitas did a good review (Freitas, 2002) of scaling-up methods for GBML, and describes a classification of these systems:

- Individual-wise: changing the subset of the training examples used for each fitness computation.
- Run-wise: selecting a static subset of examples for the whole evolutionary process.
- Generation-wise: changing the subset of the training examples used at each generation of the evolutionary process.

The run-wise approach could be seen as a prototype selection method, and the other two can be considered as *modified learning algorithms*.

For a genetic algorithms, is quite unfair to change the fitness function among individuals in the same generation. On the other hand, selecting a static subset of examples before the learning process can bias significantly the performance of the system. Therefore, most systems reported in the literature end up being generation-wise, as will be the contributions in this area that are presented in this thesis.

How can this generation-wise training examples subset be selected? Most systems reported in the literature perform a pure random sampling process to select the subset. One example of this approach, applied to attribute selection (but able to be extrapolated to rule induction), is (Sharpe & Glover, 1999). Other approaches refine the random sampling in several ways. Difficult instances (missclassified) can be used more frequently than correctly classified instances (Gathercole & Ross, 1994), or "aged" instances (instances not used for some time) can have more probability of being selected (Gathercole & Ross, 1994).

Also, in (Sharpe & Glover, 1999) the issue of what final solution is proposed by the *GA* is discussed. If the subsample used is small, maybe the best individual of the last iteration is not representative enough of the whole training set. The authors propose two approaches: the first one is to use the whole set in the last iteration. The other approach is to perform a kind of voting process among all individuals in the population.

3.6 Handling the bloat effect

The evolution of variable-length individuals can lead to solutions growing without control. This phenomenon is usually known as *bloat* (Langdon, 1997). This phenomenon can affect in general any evolutionary computation paradigm using variable-length representations, but it has been especially studied in the Genetic Programming field, where individuals are variable-length by definition. This section will describe general evolutionary computation techniques, not only GBML-related ones, because most of these techniques are easily extrapolated to GBML.

How can the control of the *bloat* effect and the generalization pressure be implemented? Several alternatives can be found in the literature. We can group most of them in three categories:

- Modification of the original fitness function to add some term related to the length of the individual
- A special selection algorithm
- Removing useless parts of the chromosome

3.6.1 Modification of the fitness function

The methods that modify the fitness function can also be grouped in two sub-categories: penalization functions and weighted sums. The control of bloat by means of a penalization function is usually known as *parsimony pressure* (Soule & Foster, 1998; Llorà, Goldberg, Traus, & Bernadó, 2002) although in a recent paper (Luke & Panait, 2002) this concept has been extended to also include all selection methods that take into account the size of the individuals. This penalization usually takes two different forms

- Multiplying the raw fitness by a penalty value proportional to the length of the individual if the length is over a certain threshold (Burke, Jong, Grefenstette, Ramsey, & Wu, 1998; Bernadó, Mekaouche, & Garrell, 1999). This method has the added difficulty of deciding which is the correct threshold, which usually is domain-specific.
- Subtracting from the raw fitness a term depending on both the length and the raw fitness (Nordin & Banzhaf, 1995; Bassett & Jong, 2000). In this way the penalty will increase gradually at the rate as the fitness, preventing a premature convergence. Again, these kind of methods usually have parameters which have to be adjusted for each domain.

The weighted sum methods can be considered a specific case of a penalization function with the particularity of having the fitness structure defined by its name. The general form of the method is described as:

$$fitness = w_1 \cdot \text{raw fitness} + w_2 \cdot \text{length} \tag{3.17}$$

One exception of this structure is (Bernadó & Garrell, 2000) which is a Michigan LCS. In this case the second term of the sum is the generality of a rule (frequency of activation) instead of the individual length. The weights of the sum can be static (Dasgupta & Gonzalez, 2001) or dynamic (Cordon, Herrera, & Villar, 2001). In the former case the domain-specific adjusting problem remains. In the latter case the individuals are fixed-length and evolve the information necessary to generate a variable-length fuzzy rules set with an heuristic algorithm. The dynamic weight is based on the population individual that generates the largest rule set.

In this category we can also add a method based on the *MDL* principle (Rissanen, 1978) applied to Genetic Programming (Iba, de Garis, & Sato, 1994). The fitness formula defined by the *MDL* principle remains a weighted sum. However, its components have a complex knowledge-dependent definition, instead of the (raw fitness,length) pair.

3.6.2 Special selection algorithms

In this category we include the control of the size or generalization pressure methods which have not aggregated the raw fitness and some other parameter (length/generalization) into a single value, thus, needing a specific selection algorithm. We can separate the methods in two categories:

- Pareto-based Multi-Objective Optimization (MOO) methods using the accuracy and either the length of the individuals (Ekart & Nemeth, 2001; Llorà, Goldberg, Traus, & Bernadó, 2002) or a generalization measure (Bernadó & Garrell, 2000) as the two objectives. The *MOLCS-GA* system (Llorà, Goldberg, Traus, & Bernadó, 2002) also introduces elitism into the selection process in order to preserve the Pareto front and best (based on accuracy) 30% of the prior population. This method, after identifying to which Pareto front belongs each individual of the population, gives to the individuals the fitness function described in equation 3.18, where I_j^i is the individual j belonging to front i, ϕ is the sharing function, using a sharing radius of 0.1, $d_{I_j^i, I_k^i}$ is the euclidean distance between individuals j and k of front i and n is the number of fronts in the population. This fitness function combines a raw multi-objective fitness (giving higher fitness to the first fronts) with a niching procedure implemented using the *sharing* method (Goldberg,

1989a) in order to spread the individuals through all the front to which they belong. After all these fitness computations are performed, tournament selection is used.

$$fitness(I_j^i) = \delta \left(\frac{1}{\sum_{k \in I^i} \phi(d_{I_j^i, I_k^i})} + n - i - 1 \right) \tag{3.18}$$

- Hierarchical Selection Algorithms. These methods (Aguirre, González, & Pérez, 2002; Luke & Panait, 2002) define an ordering of some features of the individuals and perform a multi-level Tournament selection (Goldberg & Deb, 1991) in the following way: if the first feature is better in one individual than in another the first individual is better. If the values are equal we look at the second feature, then at the third and so on. The first feature is usually the raw fitness, and the second one the length of the individuals.

3.6.3 Removing useless parts of the chromosome

In this category the systems that in certain circumstances can remove certain rules of the individuals are included. Two different approaches are described:

First, we have the *SAMUEL* (Grefenstette, 1991) system, that applies *Lamarkian* operators, that manipulate the genetic material in a deterministic way, unlike traditional GAs. This system has an operator which is activated when the size of an individual (in number of rules) is larger than a certain threshold. Some rules are eliminated based on the following criteria:

- The frequency of activation of the rule is under a given threshold
- The "strength" of the rule (a measure that combines the accuracy and the usage of the rule) is under a given threshold
- The rule has been subsumed by another rule of higher strength

On the other hand we have the *RuleDrop* operator of the *GIL* (Janikow, 1991) system described in section 3.3. This operator removes rules given a certain probability.

3.7 Summary of the chapter

This chapter has provided a description of the specific area where this thesis is focused: genetic-based machine learning (GBML). The chapter started with a general description of the base search techniques used in GBML: evolutionary computation and, specifically, genetic

algorithms (GA). After some brief theory description of GAs, the rest of the chapter focused specifically on machine learning issues. First, three models of GBML systems (and some example system of each model) were described. Finally, the chapter finished with three specific topics that are very related to the contributions presented in this thesis: representations for real-valued attributes, scaling-up of GBML systems and control of the bloat effect.

The aim of the chapter was to describe some GBML background material that allows to place the contributions of the thesis in the context of the research field where it is applied. Therefore, a lot of GBML content has been omitted or only briefly described, not because it unimportant, but because in the author's opinion it is not related to the thesis enough.

The background material section of this thesis finishes with this chapter. Next, the framework over which the contributions presented in this thesis have been built will be introduced.

Part II

Contributions to the Pittsburgh model of GBML

Chapter 4

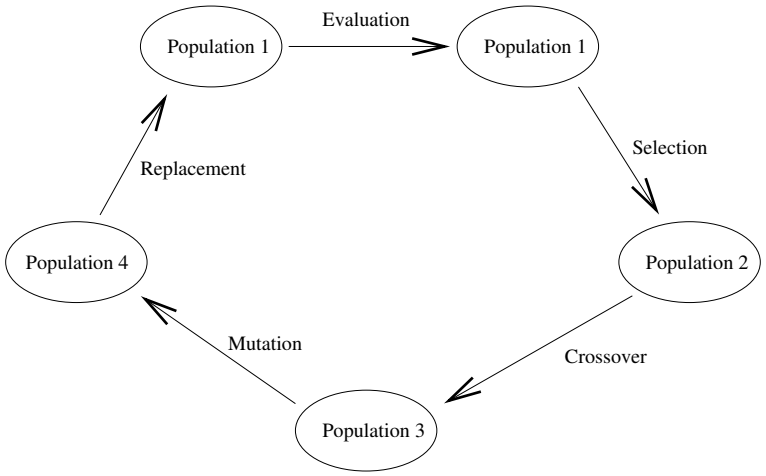
Experimental framework of the thesis

This is the first chapter of the central part of the thesis, where all the contributions to the Pittsburgh model of evolutionary learning are presented. However, before starting the description of these contributions it is necessary to describe to the maximum detail what the framework is, the basic Pittsburgh system over which these contributions are constructed. Thus, this chapter will contain all the details of the GAssist Pitt system that cannot be considered as novel contributions. These details cover issues such as the matching process, the mutation operator, random number generation, handling of missing values, etc.

This chapter will also present the experimental framework of the thesis. Usually, the datasets, statistical tests, etc. of an experimentation are described after introducing the novel contents of the thesis and (obviously) before the results. However, as each of these chapters dedicated to contributions is quite large, we have decided to report the experimentation made for each of the contributions in the same chapter where the contribution is described. This means that the experimentation framework must be described in depth before the contributions and this means placing the description of the experimentation framework in this chapter.

The chapter is structured as follows: section 4.1 will describe the general GA framework of the system, followed by section 4.2 containing the specific machine learning issues. Then, section 4.3 will describe the test suite used in the different experiments reported in the thesis, and section 4.4 will describe the experimentation methodology used. Finally, section 4.5 will include a summary of the chapter.

Figure 4.1: GA cycle used in GAssist



4.1 Framework of GAssist: general GA issues

GAssist, as a system belonging to the Pittsburgh model, uses a standard *GA* cycle, represented in figure 4.1, using the following options for each stage of the cycle:

- Evaluation: representation-dependent, detailed in next section
- Selection algorithm: tournament selection
- Mating policy for crossover: totally random
- Crossover operator: representation-dependent, detailed in next section
- Mutation probability: individual-wise. The reason for this choice is to make easier the tuning process of the GA because of the variable-length individuals that will be used.
- Mutation operator: representation-dependent, detailed in next section
- Generational replacement with elitism for the best individual

4.2 Framework of GAssist: machine learning issues

The *GAssist* system was originally inspired in *GABIL*, thus, several of the following details are common in both systems:

- Matching strategy: individuals (rule sets) will be treated as a decision list (Rivest, 1987). Therefore, the first rule that becomes true for an input instance will be used to classify it
- Fitness function: *GABIL*'s squared accuracy fitness function
- Base nominal representation: the *GABIL* representation, described in the previous chapter is the nominal representation used for nominal datasets, unless otherwise explicitly stated.
- Crossover operator: all knowledge representations used in the thesis will use the *GABIL* semantically correct crossover operator. This means selecting cut points in equivalent position inside the rule (but in any rule) for both parents
- Mutation operator: the *GABIL*'s bit-flipping mutation will be used for the nominal representation.
- Missing values policy: When we deal with datasets with missing values we use a substitution policy. That is, we gather the instances belonging to the same class as the one with missing values, and we substitute the missing value by either the most frequent value or the average value, depending on the type of attribute (nominal or real-valued).
- Pseudo-Random Numbers Generator: The pseudo-random number generator (PRNG) used is the *Mersene Twister* (Matsumoto & Nishimura, 1998), one of the best PRNGs currently available. Its application to *GAs* has been studied recently (Cantú-Paz, 2002), showing that it reduces the fluctuation of the obtained results.

4.3 Test suite of the experimentation of the thesis

A good test suite is important to legitimate an experimentation process. The datasets included should represent a broad range of possibilities in several categories:

- Number of attributes
- Number of instances
- Number of classes
- Uniform/non-uniform class distribution
- Type of attributes
- Mix of different type of attributes
- Missing values
- Synthetic and real problems
- Explicit presence of noise

The selected datasets for the experiments reported in this thesis try to represent a broad range of possibilities of the above categories. We split this list in two parts. First, the group of small datasets that are considered as *small*. These datasets will be used in all the experiments of the thesis except for the ones in chapter 7, which deals with techniques that have as an objective handling successfully large datasets. Also, not all selected datasets are used in all experiments. For instance, in chapter 6, which deals with representations for real-valued attributes, the experimentation only uses datasets where the majority of attributes are real-valued.

The origin of the datasets is also diverse. Most of them come from the University of California at Irvine (UCI) repository (Blake, Keogh, & Merz, 1998), which has become the reference experimentation framework in the machine learning community in recent years. However, in some experiments we also include datasets from the private repository of our own research group. These datasets are called *mammograms* (Martí, Cufí, Regincós, & et al., 1998), *biopsies* (Martínez Marroquín, Vos, & et al., 1996) and *learning* (Golobardes, Llorà, Garrell, Vernet, & Bacardit, 2000). A list of the small datasets (and the identifier assigned to each dataset) follows:

- Audiology (*aud*)
- Auto imports database (*aut*)
- Balance Scale Weight & Distance Database (*bal*)
- BUPA liver disorders (*bpa*)
- Biopies cancer diagnosis database (*bps*)
- Breast cancer data (*bre*)
- Contraceptive Method Choice (*cmc*)
- Horse Colic database (*col*)
- Credit Approval (*cr-a*)
- German Credit data (*cr-g*)
- Glass Identification Database (*gls*)
- Cleveland Heart Disease Database (*h-c*)
- Hepatitis Domain (*hep*)
- Hungarian Heart Disease Database (*h-h*)
- Statlog Heart Disease (*h-s*)
- Johns Hopkins University Ionosphere database (*ion*)
- Iris Plants Database (*irs*)
- Final settlements in labor negotitions in Canadian industry (*lab*)
- Learning (*lrn*)
- Lymphography Domain (*lym*)
- FIS mammogram database (*mmg*)
- Pima Indians Diabetes Database (*pim*)
- Primary Tumor Domain (*prt*)
- Sonar, Mines vs. Rocks (*son*)

- Large Soybean Database (*soy*)
- Thyroid gland data (*thy*)
- Vehicle silhouettes (*veh*)
- 1984 United States Congressional Voting Records Database (*vot*)
- Wisconsin Breast Cancer Database (*wbcd*)
- Wisconsin Diagnostic Breast Cancer (*wdbc*)
- Wine recognition data (*wine*)
- Wisconsin Prognostic Breast Cancer (*wpbc*)
- Zoo database (*zoo*)

Some of these datasets have specific citation requests: The audiology dataset was donated by Professor Jergen at Baylor College of Medicine. The breast cancer, lymphography, and primary tumor domains were obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data. The Cleveland Heart Disease database was donated by Robert Deytrano, M.D., Ph.D. from the Cleveland Clinic Foundation. The Hungarian Heart Disease database was donated by Andras Janosi, M.D. from the Hungarian Institute of Cardiology. The Vehicle silhouettes datasets comes from the Turing Institute, Glasgow, Scotland. The Wisconsin breast cancer databases were obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg (Wolberg & Mangasarian, 1990).

As said before, in chapter 7 a different set of problems, beside these ones, will be used. The origin of the problems is also diverse. First of all, there are two synthetic problems: *Multiplexer* and *led*. The first one is actually a family of problems, widely used in the Learning Classifiers Systems field (Wilson, 1995). It consist in learning the transition table of a multiplexer. Depending on the number of data inputs of the multiplexer we can speak of MX-6, with 6 attributes and 64 instances, MX-11 with 11 attributes and 2048 instances, ... The other synthetic dataset is called *LED*. The problem consists of learning the digit represented by a seven-segments display. This problem comes from the *UCI* repository and it is actually a generator. It can generate an arbitrary number of instances of the domain with a given percentage of noise applied to the attributes. The standard 10% noise level will be used here.

The rest of the problems used are the ones that can be considered as medium-size or large. The first or them, called *FARS* (Fatality Analysis Reporting System), is a compilation

Table 4.1: Features of the small datasets used in this thesis. #Inst. = Number of Instances, #Attr. = Number of attributes, #Real = Number of real-valued attributes, #Nom. = Number of nominal attributes, #Cla. = Number of classes, Dev.cla. = Deviation of class distribution, Maj.cla. = Percentage of instances belonging to the majority class, Min.cla. = Percentage of instances belonging to the minority class, MV Inst. = Percentage of instance with missing values, MV Attr. = Number of attributes with missing values, MV values = Percentage of values ($\#instances \cdot \#attr$) with missing values

	Dataset Properties										
Domain	#Inst.	#Attr.	#Real	#Nom.	#Cla.	Dev.cla.	Maj.cla.	Min.cla.	MV Inst.	MV Attr.	MV values
aud	226	69	—	69	24	6.43%	25.22%	0.44%	98.23%	7	2.00%
aut	205	25	15	10	6	10.25%	32.68%	1.46%	22.44%	7	1.11%
bal	625	4	4	—	3	18.03%	46.08%	7.84%	—	—	—
bpa	345	6	6	—	2	7.97%	57.97%	42.03%	—	—	—
bps	1027	24	24	—	2	1.60%	51.61%	48.39%	—	—	—
bre	286	9	—	9	2	20.28%	70.28%	29.72%	3.15%	2	0.31%
cmc	1473	9	2	7	3	8.26%	42.70%	22.61%	—	—	—
col	368	22	7	15	2	13.04%	63.04%	36.96%	98.10%	21	22.77%
cr-a	690	15	6	9	2	5.51%	55.51%	44.49%	5.36%	7	0.61%
cr-g	1000	20	8	12	2	20.00%	70.00%	30.00%	—	—	—
gls	214	9	9	—	6	12.69%	35.51%	4.21%	—	—	—
h-c	303	13	6	7	2	4.46%	54.46%	45.54%	2.31%	2	0.17%
hep	155	19	6	13	2	29.35%	79.35%	20.65%	48.39%	15	5.39%
h-h	294	13	6	7	2	13.95%	63.95%	36.05%	99.66%	9	19.00%
h-s	270	13	13	—	2	5.56%	55.56%	44.44%	—	—	—
ion	351	34	34	—	2	14.10%	64.10%	35.90%	—	—	—
irs	150	4	4	—	3	—	33.33%	33.33%	—	—	—
lab	57	16	8	8	2	14.91%	64.91%	35.09%	98.25%	16	33.64%
lrn	648	6	4	2	5	14.90%	45.83%	1.54%	—	—	—
lym	148	18	3	15	4	23.47%	54.73%	1.35%	—	—	—
mmg	216	21	21	—	2	6.01%	56.02%	43.98%	—	—	—
pim	768	8	8	—	2	15.10%	65.10%	34.90%	—	—	—
prt	339	17	—	17	21	5.48%	24.78%	0.29%	61.06%	5	3.69%
son	208	60	60	—	2	3.37%	53.37%	46.63%	—	—	—
soy	683	35	—	35	19	4.31%	13.47%	1.17%	17.72%	34	9.50%
thy	215	5	5	—	3	25.78%	69.77%	13.95%	—	—	—
veh	846	18	18	—	4	0.89%	25.77%	23.52%	—	—	—
vot	435	16	—	16	2	11.38%	61.38%	38.62%	46.67%	16	5.30%
wbcd	699	9	9	—	2	15.52%	65.52%	34.48%	2.29%	1	0.23%
wdbc	569	30	30	—	2	12.74%	62.74%	37.26%	—	—	—
wine	178	13	13	—	3	5.28%	39.89%	26.97%	—	—	—
wpbc	198	33	33	—	2	26.26%	76.26%	23.74%	2.02%	1	0.06%
zoo	101	16	—	16	7	11.82%	40.59%	3.96%	—	—	—

of statistics about car accidents made by the U.S. National Center for Statistics and Analysis [1]. The specific dataset used contains information about all people involved in car accidents in the U.S. during 2001. The selected class is the level of injury suffered. Finally, the rest of datasets are real datasets that come from the *UCI* repository. The list of datasets, with their features detailed in table 4.2 follows:

- Adult (*adu*)
- Connect-4 (*c-4*)
- FARS (*fars*)
- Hypothyroid (*hyp*)
- King-rook-vs-king-pawn (*krkp*)
- Mushroom (*mush*)
- Nursery (*nur*)
- Pen-Based Recognition of Handwritten Digits (*pen*)
- Statlog - Landsat Satellite (*sat*)
- Segment (*seg*)
- Sick (*sick*)
- Splice (*spl*)
- Waveform (*wav*)

4.4 Experimentation methodology

Once the test suite is selected, it is time to decide how the experiments will be done and how the results of these experiments will be analyzed. The aim of the experimentation design is to estimate the performance of the learning system and configuration being tested. In order to achieve this objective, some partitions of the each dataset into training and test sets is proposed, together with a formula to estimate the accuracy of the learning system based on its performance on each pair of training/test sets.

[1] Downloaded from `ftp://ftp.nhtsa.dot.gov/FARS/`

Table 4.2: Features of the large datasets used in this thesis. #Inst. = Number of Instances, #Attr. = Number of attributes, #Real = Number of real-valued attributes, #Nom. = Number of nominal attributes, #Cla. = Number of classes, Dev.cla. = Deviation of class distribution, Maj.cla. = Percentage of instances belonging to the majority class, Min.cla. = Percentage of instances belonging to the minority class, MV Inst. = Percentage of instance with missing values, MV Attr. = Number of attributes with missing values, MV values = Percentage of values ($\#instances \cdot \#attr$) with missing values

	Dataset Properties										
Domain	#Inst.	#Attr.	#Real	#Nom.	#Cla.	Dev.cla.	Maj.cla.	Min.cla.	MV Inst.	MV Attr.	MV values
adu	48842	14	6	8	2	26.07%	76.07%	23.93%	7.41%	3	0.95%
c-4	67557	42	0	42	3	23.79%	65.83%	9.55%	0.00%	0	0.00%
fars	100968	29	5	24	8	13.08%	41.71%	0.01%	0.00%	0	0.00%
hyp	3772	29	7	22	4	38.89%	92.29%	0.05%	100.00%	8	5.54%
krkp	3196	36	0	36	2	2.22%	52.22%	47.78%	0.00%	0	0.00%
mush	8124	22	0	22	2	1.80%	51.80%	48.20%	30.53%	1	1.39%
nur	12960	8	0	8	5	15.33%	33.33%	0.02%	0.00%	0	0.00%
pen	10992	16	16	0	10	0.40%	10.41%	9.60%	0.00%	0	0.00%
sat	6435	36	36	0	6	6.19%	23.82%	9.73%	0.00%	0	0.00%
seg	2310	19	19	0	7	0.00%	14.29%	14.29%	0.00%	0	0.00%
sick	3772	29	7	22	2	43.88%	93.88%	6.12%	100.00%	8	5.54%
spl	3190	60	0	60	3	13.12%	51.88%	24.04%	0.00%	0	0.00%
wav	5000	40	40	0	3	0.36%	33.84%	33.06%	0.00%	0	0.00%

The selected methodology is the most widely used in the literature: stratified ten-fold cross validation (Kohavi, 1995). In short, this method splits the dataset into 10 mutually exclusive subsets (called *folds*) of approximately the same size that also have the same class distribution existing in the whole dataset. Once the folds are created, it generates ten pairs of training/test sets. The first one uses the 9 first folds as training and the 10th one as test. The second pair uses folds 1-8 and fold 10 for training, and fold 9 for test, and so on. In order to reduce the bias that might be created by the random creation of these folds, in the experimentation of the thesis 3 sets of cross-validation folds will be used. Also, for all the stochastic systems included in the experiments, 5 runs with different random seeds will be made for each pair of training/test sets. This means that the accuracy obtained from each method is the average of 150 runs.

Once the experiments are done, and we obtain an estimate of the accuracy of each system based on the cross-validation methodology, it will be time to compare the differences between the tested systems. In order to guarantee that solid conclusions can be extracted from these comparisons it is necessary to use statistical tests. The chosen test is the Student paired t-test (Goulden, 1956). When more than two systems are included in the comparison, the Bonferroni correction (Shaffer, 1995) is used. Confidence level is set to 95%. All the tests are performed using the *R* statistical package (Venables & Ripley, 2002).

4.5 Summary of the chapter

The chapter focused on describing the framework specifically necessary to build the contributions to the Pittsburgh model presented in the four next chapters of this thesis. This framework had two parts. The first one was a a description of the basic pieces of the learning system used in the thesis: *GAssist*. The second one was the design of the experimentation framework used in this thesis to validate the contributions presented. The experimentation consists in a dataset base with a very large range of problems, to be able to extract useful information about what are the strong points or weak points of the classifier. Also, the results of experimenting with these datasets will be analyzed with statistical t-tests, to guarantee that we can extract solid conclusions of the results.

One the description of this framework is finished, it is time to really start the central part of the thesis, describing the novel contributions made.

Chapter 5

Integrating an explicit and static default rule in the Pittsburgh model

An interesting feature of encoding the individuals of a Pittsburgh Learning Classifier System as a decision list is the emergent generation of a default rule. With a default rule we can generate more compact and accurate rule sets. However, the performance of the system is strongly tied to the learning system choosing the correct class for this default rule. This chapter describes the research that has been done on extending the knowledge representation used in GAssist with an explicit and static default rule, and the policies studied to choose the correct default class.

The chapter is structured as follows: First, section 5.1 will show a brief introduction to the rest of the chapter and the motivation of this research. Then, section 5.2 will describe some background material and related work. Section 5.3 will report the modifications applied to the knowledge representation of the system to integrate the default rule. Section 5.4 will show some illustrative results of the simple policies. After the simple policies, we will describe more sophisticated ones in Section 5.5. Section 5.6 will show the experimentation results of applying the previous described policies and will also analyze in depth these results. Section 5.7 will discuss these results and will propose some further work. Finally, Section 5.8 will summarize the chapter.

Figure 5.1: Unordered and ordered rule sets for the MX-11 domain

Unordered MX-11 rule set
0 0 0 0 # # # # # # # # : 0
0 0 0 1 # # # # # # # # : 1
0 0 1 # 0 # # # # # # # : 0
0 0 1 # 1 # # # # # # # : 1
0 1 0 # # 0 # # # # # # : 0
0 1 0 # # 1 # # # # # # : 1
0 1 1 # # # 0 # # # # # : 0
0 1 1 # # # 1 # # # # # : 1
1 0 0 # # # # 0 # # # # : 0
1 0 0 # # # # 1 # # # # : 1
1 0 1 # # # # # 0 # # # : 0
1 0 1 # # # # # 1 # # # : 1
1 1 0 # # # # # # 0 # # : 0
1 1 0 # # # # # # 1 # # : 1
1 1 1 # # # # # # # 0 # : 0
1 1 1 # # # # # # # 1 # : 1

Ordered MX-11 rule set
0 0 0 0 # # # # # # # # : 0
0 0 1 # 0 # # # # # # # : 0
0 1 0 # # 0 # # # # # # : 0
0 1 1 # # # 0 # # # # # : 0
1 0 0 # # # # 0 # # # # : 0
1 0 1 # # # # # 0 # # # : 0
1 1 0 # # # # # # 0 # # : 0
1 1 1 # # # # # # # 0 # : 0
: 1

5.1 Introduction and motivation

Default rules can be very useful in combination with a decision list because the size of the rule set can be reduced significantly. For instance, for the 11-bit multiplexer we can obtain a rule set of 9 rules instead of 16 unordered ones, as represented in Figure 5.1. With a smaller rule set, the search space is reduced resulting in two potential advantages: (1) the learner has to learn less rules (representing only the other classes of the dataset) and (2) with a smaller rule set the system may be less sensitive to over-learning potentially increasing the test accuracy of the system.

Data Mining problems can also benefit from a default rule. To illustrate this, table 5.1 shows the results of running the GAssist system with no static default rule for the *Glass* problem from the *UCI* repository (Blake, Keogh, & Merz, 1998). The settings of the system are summarized in table 5.2. The partitioning of the dataset into training and test subsets is done using the stratified ten-fold cross validation method, and tests for each fold are repeated a hundred times with different random seeds. The results show the benefits of using a default rule and, more importantly, the benefits of choosing the correct class for the default rule.

Table 5.1: How the emergent generation of a default rule can affect the performance in the *Glass* dataset

Runs generating a default rule	736
Runs not generating a default rule	264
Accuracy of runs with a default rule	66.98±8.00
Accuracy of runs without a default rule	66.27±7.79
Average accuracy of runs using class 1 as default rule	65.45±7.39
Average accuracy of runs using class 2 as default rule	67.76±7.81
Average accuracy of runs using class 3 as default rule	59.40±5.51
Average accuracy of runs using class 4 as default rule	66.18±8.70
Average accuracy of runs using class 5 as default rule	67.66±8.58
Average accuracy of runs using class 6 as default rule	64.48±7.36

Table 5.2: Settings of GAssist for the default rule tests

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	300
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Number of strata of ILAS windowing	2
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Probability of Reinitialize (begin,end)	(0.02,0)
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of active rules + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

5.2 Background material and related work

We can find previous uses of a static default rule in the *LCS* field, although not in an explicit way: Classic Pitt-approach systems such as *GABIL* (DeJong, Spears, & Gordon, 1993) or GIL (Janikow, 1991), which perform concept learning (learning a concept from sets of positive/negative examples), implicitly have a default rule that covers the negative examples. The rules generated do not have an associated class because all of them cover the positive examples. However, there is no explicit policy to decide which set is the positive or negative in order to learn better. The decision comes from the definition of the dataset.

Looking at the machine learning field in general we find other examples of default rules. The C4.5rules system (Quinlan, 1993) uses an explicit default rule and like our system it generates a rule set acting as a decision list. To select the class for this default rule it uses the class that has less instances covered by the other rules in the rule set. This kind of approach seems feasible when we have induced the rule set beforehand, instead of using it during learning as our system does.

The *IREP* system (Cohen, 1995) induces, in order, the rules modeling each class of the problem (using the instances of the classes still to be learned as negative examples). The criteria of this global order is ascendant frequency of examples. Therefore, the default rule of this system uses a majority class policy. The *AQ15* (Michalski, Mozetic, & Hong, 1986) and *CN2* (Clark & Niblett, 1989) systems also use a majority class policy.

5.3 The static default rule mechanism

The requirements of the system to be able to use this mechanism are few: we only need to codify our individuals as decision list, independent on the knowledge representation used. The implementation of the static default rule is very simple. Basically it affects only the matching function, that classifies an input instance using the default class if no rule matches the instance, represented by the code in Figure 5.2. Also, the default class is removed from the classes that can be used by the rest of the rules in the population, effectively reducing the search space. A general representation of the extended rule set is shown in Figure 5.3. These changes and other small details are summarized as follows:

1. We determine with some criterion (in the following sections several criteria are studied) which class is the default class

2. An individual predicts this default class when no rule matches an input instance

Figure 5.2: Match process using an static default rule

Match process
Input : $RuleSet, Instance$
$Index = 0$
$Found = false$
While $Index < RuleSet.size$ and not $Found$ **Do**
  If $RuleSet.rule[Index]$ matches $Instance$ **Then**
    $Class = RuleSet.rule[Index].class$
    $Found = true$
  Else
    $Index++$
  EndIf
EndWhile
If not $Found$ **Then**
  $Class = DefaultClass$
EndIf
Output : Predict class $Class$ for instance $Instance$

3. The other rules of the individual cannot use the default class. Neither initialization nor mutation can make a regular rule of the individual point to the default class

4. The default rule is included in the size of the rule set. This means that the rest of the system transparently see an individual with one more rule. This affects the parts of the fitness formula that uses the size of the rule set as a variable

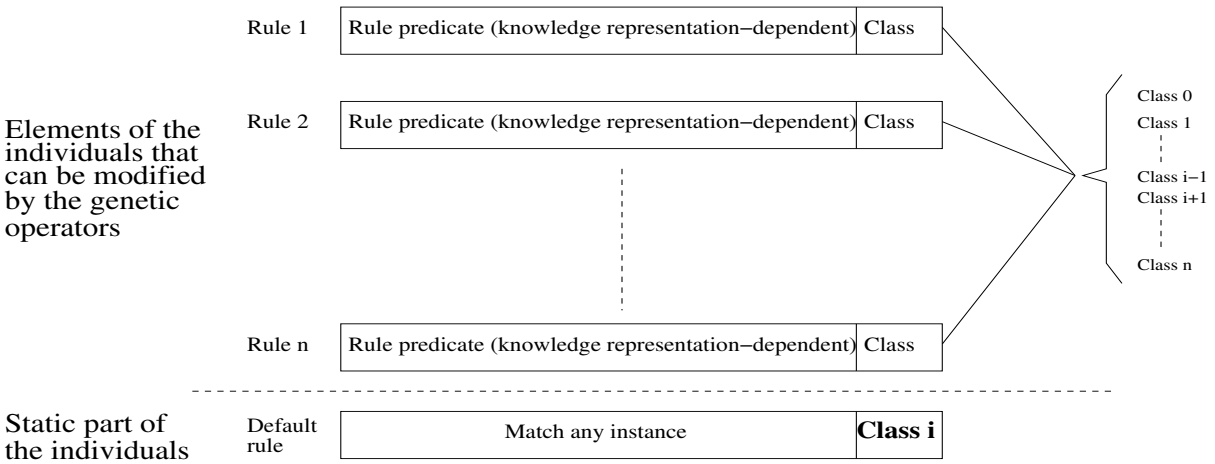
5. The default rule cannot be affected by crossover not mutation nor any other recombination operator

6. The rule deletion operator ignores the petitions to delete this rule, in the rare chances that this rule matches nothing

7. The MDL-based fitness function computes a theory length for this rule supposing that the rule is totally general, that is, as if it were the emergent default rule observed before implementing this mechanism

For the specific case of two-class domains, the classification problem is transformed into a concept learning problem and the resulting knowledge representation is quite close to the ones used in other evolutionary concept learning systems like *GABIL* (DeJong, Spears, & Gordon, 1993) or *GIL* (Janikow, 1991).

Figure 5.3: Representation of the extended rule set with the static default rule



5.4 Simple policies to determine the default class

In order to answer the question of which class is suitable for being the default class we start by experimenting with two simple policies: using the most and least frequent class in the domain (majority and minority classes). In Section 5.6 we can see the results of these tests for several datasets. Here we show the results (in Table 5.3 only of two datasets (*Glass* and *Ionosphere*), also from UCI. For *Glass* the best policy is using the majority class. For *Ionosphere* the best policy is using the minority class. The point of showing these two datasets is that it is very difficult to decide *a priori* which is the most suitable default rule class for each dataset. Also, we can see in the values of the training accuracy and the number of rules a hint about how we can combine the two policies to maximize the performance of the system. In Section 5.6 we show a simple combination consisting of choosing at the test stage the policy which has more training accuracy.

5.5 Automatically determined default class

Given that neither the majority nor the minority policies are always the most suitable policy as default class, the next step is to modify the system to automatically determine the best default class. Our initial approach simply assigns a randomly chosen class as default class for each individual in the initial population. Additionally, we introduce a restricted mating

Table 5.3: Results using majority and minority policy for the default class in the *Glass* and *Ionosphere* datasets.

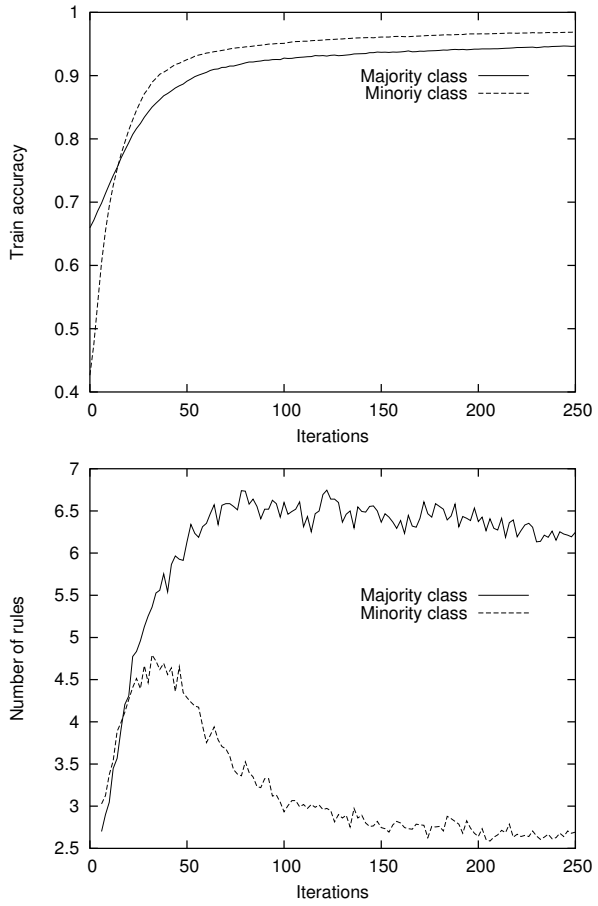
Domain	Def. Class. Policy	Training accuracy	Test accuracy	Number of rules
Glass	disabled	79.9±2.6	66.4±8.1	6.4±0.7
Glass	majority	83.2±1.6	69.5±6.9	6.6±0.8
Glass	minority	80.6±2.3	66.7±8.0	7.2±0.8
Ionosphere	disabled	96.0±0.6	92.8±3.6	2.3±0.6
Ionosphere	majority	95.7±0.8	90.0±4.4	5.7±1.2
Ionosphere	minority	96.8±0.7	93.0±3.7	2.6±0.8

Figure 5.4: Code of the crossover algorithm with restricted mating

Niched crossover algorithm
Comment To simplify the code, $Parents$ contains only the parent individuals
Comment already selected for crossover by the probability of crossover
Input : $Parents$
$OffspringSet = \emptyset$
While $Parents$ is not empty
    $Parent1$ = select randomly and individual from $Parents$
    Remove $Parent1$ from $Parents$
    $Niche$ = default class of $Parent1$
    If there are individuals in $Parents$ belonging to $Niche$
        $Parent2$ = select randomly and individual from $Parents$ belonging to $Niche$
        Remove $Parent2$ from $Parents$
        $Offspring1, Offspring2$ = apply crossover to $Parent1, Parent2$
        Add $Offspring1, Offspring2$ to $OffspringSet$
    Else
        $Offspring$ = clone of $Parent1$
        Add $Offspring$ to $OffspringSet$
    EndIf
EndWhile
Output : $OffspringSet$

mechanism to avoid crossover operations between individuals having different default classes, summarized by the code in Figure 5.4. Having removed the default class from the rest of the rules, crossing individuals with different default classes may create lethals with high probability. Especially in the specific case of two-classes domains, the regular rules of individuals using different default classes cover completely different subsets of rules, therefore it is impossible to integrate the rules of these two individuals using the regular crossover operator.

Figure 5.5: Evolution of the training accuracy and the number of rules for the Ionosphere problem using majority/minority default class policies



If we run the system in this setting, we mainly observe that all individuals with one default class take over the population. The question is can the system choose the correct default class during the initial iterations?. To answer this question, we show the evolution of the training accuracy and the number of rules for the *Ionosphere* tests described in Figure 5.5. We can see that the training accuracy of the default class policy using the suitable class for this problem (that is, the minority class) is lower at the initial iterations than the accuracy of the majority class policy. Also, we can see the reason for the better test accuracy of the minority policy in the smaller (better generalized) rule set created by this policy.

Thus, it appears necessary to introduce an additional niching mechanism that preserves individuals for all default classes until the system has learned enough to decide correctly on the best default class. This niching is achieved using a modified tournament selection mechanism,

Figure 5.6: Code for the niched tournament selection

Niched tournament selection
Input : $Population, PopSize, NumNiches, TournamentSize$
$NextPopulation = \emptyset$
For $i = 1$ to $NumNiches$
    $ProportionNiche[i] = PopSize/NumNiches$
EndFor

For $i = 1$ to $PopSize$
    $Niche =$ select randomly a niche based on ProportionNiche
    $ProportionNiche[Niche] - -$
    Select $TournamentSize$ individuals from $Population$ belonging to $Niche$
    $winner$=Apply tournament
    Add $winner$ to $NextPopulation$
EndFor
Output : $NextPopulation$

inspired in (Oei, Goldberg, & Chang, 1991) in which the individuals participating in each tournament are forced to belong to the same class. Also, each default class has an equal number of tournaments. This niched tournament selection is represented by the code in Figure 5.6. The tournament with niche preservation is used until the best individuals of each default class have similar training accuracy. After this point, the niching is disabled and the system chooses freely among the individuals. Specifically, we compute for each niche the average accuracy for the last 15 iterations of its best individual. When the standard deviation of all these averages is smaller than 0.5%, we disable the niched tournament selection.

Summarizing, the changes introduced to the default rule model by the automatic policy are the following:

1. Initialization assigns randomly to each individual a class as being the default class
2. Again, this class cannot be used in the regular rules of the individual
3. Individuals having different default class cannot cross among them. The crossover algorithm is modified adding this mating restriction.
4. We use a niched tournament selection to preserve an uniform proportion of individuals from all default classes in the population. This niching process is achieved reserving a quota of tournaments to each niche, and only applying tournaments among individuals belonging to the same niche.

5. This niching mechanism is disabled when individuals using different default class can compete fairly among themselves. Specifically, we compute, for each default class, the average accuracy for the last 15 iterations of its best individual. When the standard deviation of all these averages is smaller than 0.5%, the niched tournament selection is disabled and a regular tournament selection takes places until the end of the learning process.

5.6 Results

In this section, we show the results of comparing the three policies tested for the default class (*majority,minority,auto* to the original system (*orig*) with emergent default rule. We also test a fifth configuration (*majority+minority*): Choosing for the test stage the majority/minority policy that obtained more training accuracy. This configuration usually chooses the correct policy (although there are some exceptions, like *bpa*). However, this last configuration takes twice as long as the other policies.

The tests include 15 datasets (*bpa, bps, gls, h-s, ion, lrn, mmg, pim, son, thy, veh, wdbc, wbcd, wine, wpbc*) described in section 4.3. Table 5.4 shows the results for these tests. These results were analyzed using pair-wise statistical t-tests with Bonferroni correction to determine how many times each method could significantly outperform or be outperformed by the other methods. These statistical tests are summarized in table 5.5.

At first glance, we can see that all but two datasets (*wbcd* and *wpbc*) can benefit (by one or more of the studied default class policies) from the inclusion in the knowledge representation of a default rule. However, the achieved accuracy increase is not uniform across the datasets. Some of them, like *gls* or *son*, show a notable accuracy increase, while some others only show a small, non-significant increase. To understand these different degrees of accuracy increase we have computed the percentage of runs where the *orig* configuration was already generating emergently a default rule. Table 5.6 shows these results and also the accuracy of the *orig* configuration and also the accuracy of the best default class policy for each dataset (and their difference). Although it is not totally clear, we can see correlation between the percentage of discovered default rules and the accuracy difference between using/not using the default rule. The clearest exception is the *gls* dataset. However, considering that this dataset has 6 classes, the benefits of removing the default class from the pool of classes used in the regular rules are already substantial even if the *orig* configuration was already using a default rule.

From the test accuracy averages and the t-tests results it is clear that the *major+minor* policy is the best configuration, both in performance and robustness, because it has been never

Table 5.4: Results of the tests comparing the studied default class policies to the original configuration using pop. size 300

Domain	Result	Default rule policy				
		Disabled	Major	Minor	Auto	Major+Minor
bpa	Training acc.	78.6±1.6	81.4±1.3	80.1±1.6	80.8±1.4	81.4±1.3
	Test acc.	63.8±7.4	62.9±7.8	65.2±6.5	64.0±6.9	62.9±7.8
	#rules	6.7±1.0	8.9±1.4	8.3±1.5	8.5±1.6	8.9±1.4
bps	Training acc.	84.8±0.9	86.0±0.7	86.8±0.7	86.6±0.7	86.8±0.7
	Test acc.	80.1±3.9	81.2±3.6	81.5±3.6	81.4±3.7	81.5±3.6
	#rules	5.1±0.4	6.1±1.1	5.7±0.9	5.6±0.8	5.7±0.9
gls	Training acc.	79.9±2.6	83.2±1.6	80.6±2.3	79.0±1.8	83.2±1.6
	Test acc.	66.4±8.1	69.5±6.9	66.7±8.0	66.9±7.4	69.5±6.9
	#rules	6.4±0.7	6.6±0.8	7.2±0.8	6.9±0.9	6.6±0.8
h-s	Training acc.	89.8±1.2	91.6±0.9	92.1±0.8	91.9±0.9	92.1±0.8
	Test acc.	79.5±6.2	79.3±6.4	81.3±6.8	81.3±6.1	81.3±6.8
	#rules	6.7±0.9	7.6±1.2	7.3±1.2	7.4±1.3	7.3±1.2
ion	Training acc.	96.0±0.6	95.7±0.8	96.8±0.7	96.8±0.7	96.8±0.7
	Test acc.	92.8±3.6	90.0±4.4	93.0±3.7	93.1±3.9	93.0±3.7
	#rules	2.3±0.6	5.7±1.2	2.6±0.8	2.6±0.7	2.6±0.8
lrn	Training acc.	75.2±1.9	76.8±0.8	75.4±1.4	75.4±1.0	76.8±0.8
	Test acc.	68.5±4.7	68.9±5.7	68.9±4.5	68.6±5.6	68.9±5.7
	#rules	8.5±1.9	9.6±1.9	9.2±1.9	8.6±1.7	9.6±1.9
mmg	Training acc.	79.7±1.8	83.2±1.3	83.1±1.3	83.0±1.4	83.2±1.3
	Test acc.	66.2±7.8	68.9±8.3	67.8±8.4	66.8±9.0	68.9±8.3
	#rules	6.5±0.8	6.7±0.9	6.7±0.8	6.6±0.9	6.7±0.9
pim	Training acc.	79.7±0.9	81.3±0.8	80.9±0.7	81.1±0.8	81.3±0.8
	Test acc.	74.7±4.7	75.4±4.8	75.0±4.7	75.0±4.5	75.4±4.8
	#rules	5.2±0.4	6.2±1.0	5.6±0.8	6.1±1.0	6.2±1.0
son	Training acc.	92.2±1.6	96.1±1.2	94.8±1.4	95.5±1.4	96.1±1.2
	Test acc.	72.6±11.5	77.0±9.0	76.1±9.7	76.1±9.3	77.0±9.0
	#rules	6.7±1.1	7.6±1.4	7.7±1.3	7.4±1.1	7.6±1.4
thy	Training acc.	97.4±1.0	98.4±0.7	98.4±0.7	98.1±0.8	98.4±0.7
	Test acc.	91.9±5.6	92.8±4.8	92.3±5.3	92.2±5.6	92.8±4.8
	#rules	5.2±0.4	5.7±0.6	5.4±0.5	5.5±0.6	5.7±0.6
veh	Training acc.	71.1±2.2	73.5±1.4	73.5±1.4	72.0±1.5	73.5±1.4
	Test acc.	66.4±4.7	68.1±4.5	67.4±4.9	67.5±4.7	68.1±4.5
	#rules	6.6±1.2	9.3±2.0	9.9±1.6	8.0±1.8	9.3±2.0
wbcd	Training acc.	97.7±0.3	98.2±0.3	98.4±0.3	98.4±0.3	98.4±0.3
	Test acc.	95.9±2.2	95.0±2.5	95.7±2.0	95.6±2.2	95.7±2.0
	#rules	2.6±0.7	5.8±1.2	3.2±0.6	3.3±0.7	3.2±0.6
wdbc	Training acc.	97.2±0.8	97.8±0.6	97.8±0.6	97.8±0.7	97.8±0.6
	Test acc.	94.1±3.0	94.2±3.1	94.0±3.0	94.3±3.1	94.2±3.1
	#rules	4.3±1.1	4.6±0.9	4.4±1.0	4.5±1.0	4.6±0.9
wine	Training acc.	99.4±0.5	99.7±0.4	99.9±0.3	99.6±0.4	99.9±0.3
	Test acc.	92.7±5.9	93.3±6.2	92.2±6.3	93.9±5.9	92.2±6.3
	#rules	3.8±0.7	3.6±0.6	4.1±0.5	3.8±0.6	4.1±0.5
wpbc	Training acc.	84.3±3.0	89.4±2.0	86.4±3.4	88.7±2.3	89.4±2.0
	Test acc.	76.0±7.3	75.8±7.4	72.6±8.5	75.2±7.5	75.8±7.4
	#rules	2.8±0.8	3.8±0.9	4.2±1.2	3.6±1.0	3.8±0.9
ave.	Training acc.	86.9±9.0	88.8±8.4	88.3±8.8	88.3±9.0	89.0±8.5
	Test acc.	78.8±11.4	79.5±10.7	79.3±11.0	79.5±11.3	79.8±10.9
	#rules	5.3±1.8	6.5±1.8	6.1±2.1	5.9±1.9	6.1±2.1

Table 5.5: Summary of the statistical t-tests applied to the results of the default rule experimentation with a population size of 300 and using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

Policy	Disabled	Major	Minor	Auto	Major+Minor	Total
Disabled	-	2	1	0	0	3
Major	3	-	2	1	0	6
Minor	2	2	-	0	0	4
Auto	2	1	1	-	0	4
Major+Minor	4	2	2	1	-	9
Total	11	7	6	2	0	

Table 5.6: Percentage of runs where *orig* configuration was already generating a default rule, accuracy difference between *orig* and the best default class policy for each dataset.

Rows are sorted by the percentage of default rule generation in *orig*

Label	meaning
DRG	Percentage of runs where the default rule was generated in *orig* configuration
AccO	Accuracy of the *orig* configuration
AccDR	Accuracy of the best rule policy on the dataset
AccDif	Accuracy difference between AccO and AccDR

Dataset	DRG	AccO	AccDR	AccDif
mmg	19.33%	66.21%	68.88%	-2.67%
son	36.00%	72.58%	76.99%	-4.42%
bps	40.00%	80.10%	81.55%	-1.44%
veh	46.67%	66.43%	68.15%	-1.72%
pim	50.67%	74.65%	75.37%	-0.71%
wdbc	55.33%	94.06%	94.26%	-0.20%
h-s	57.33%	79.46%	81.31%	-1.85%
bpa	65.33%	63.79%	65.22%	-1.43%
thy	68.67%	91.92%	92.79%	-0.87%
wine	71.33%	92.74%	93.85%	-1.12%
gls	74.00%	66.37%	69.52%	-3.15%
lrn	76.00%	68.55%	68.93%	-0.39%
wpbc	82.00%	76.03%	75.78%	0.25%
ion	86.00%	92.85%	93.13%	-0.29%
bre	96.00%	95.88%	95.74%	0.14%

Table 5.7: Default class behavior in the auto configuration

Dataset	Major. class pos.	Minor. class pos.	Class distribution in default rule
bpa	2	1	(50.67%,49.33%)
bps	1	2	(14.67%,85.33%)
bre	1	2	(0.00%,100.00%)
gls	2	4	(14.00%,40.00%,8.67%,9.33%,14.00%,14.00%)
h-s	1	2	(32.00%,68.00%)
ion	2	1	(97.33%,2.67%)
lrn	1	5	(17.33%,35.33%,34.00%,11.33%,2.00%)
mmg	1	2	(48.00%,52.00%)
pim	1	2	(62.00%,38.00%)
son	2	1	(32.00%,68.00%)
thy	1	3	(40.67%,18.67%,40.67%)
veh	3	4	(35.33%,24.00%,13.33%,27.33%)
wdbc	2	1	(48.00%,52.00%)
wine	2	3	(4.00%,70.67%,25.33%)
wpbc	2	1	(1.33%,98.67%)

outperformed in a significant way. However, having in this configuration a run-time two times larger than in the other configurations, we have to question whether the computational cost sacrifice is worth it. Looking at the other configurations, *major* and *auto* are tied in accuracy average, but *auto* is much more robust than *major* according to the t-tests.

Nevertheless, it is important to investigate why the *auto* policy presents lower performance than *major+minor*. Table 5.7 shows the class distribution of the default rules that appear in the *auto* configuration runs, and we can see that this configuration is not able to determine always which is the most suitable default class. Actually, on only 5 of the 15 datasets the chosen default class was almost or totally concentrated on a single class.

Another important issue is the number of iterations where the niched tournament selection was used. Table 5.8 shows these results. We can see that for some datasets, the niching process was used for quite a long time. It is reported in the niching literature (Goldberg, 1989b) that we should increase the population size in order to guarantee that all niches can learn properly.

For this reason, a second set of tests was performed increasing the population size from 300 to 400. The results are shown in table 5.9. The summary of the statistical t-tests applied to these results is in table 5.10.

Now we can see a different scenario. The increase in population size allows the *auto* policy to permit all niches to learn properly. This fact is reflected on the accuracy performance of this policy which manages to reach *major+minor*, both in accuracy and in robustness, based on the

Table 5.8: Percentage of iterations that used the niched tournament selection in the default rule auto configuration

Dataset	Percentage of iterations
bpa	8.19%
bps	15.10%
bre	13.71%
gls	27.82%
h-s	13.33%
ion	6.72%
lrn	69.06%
mmg	10.79%
pim	9.41%
son	15.45%
thy	30.20%
veh	20.29%
wdbc	7.66%
wine	34.11%
wpbc	12.43%

t-tests. Now that both policies are competitive, the smaller computational cost of *auto* (also compared to *major+minor* using a population size of 300) clearly makes it the most suitable configuration for the default class.

Also, we can see how the only method that degrades performance when we increase the population size is the majority class policy, suggesting that the system is sensitive to over-learning in domains where the majority class policy is not suitable. The larger average number of rules and the better training accuracy of the solutions generated by this policy confirm the over-learning problem compared to all other non-composed policies.

5.7 Discussion and further work

One of the main sacrifices done in the *auto* default class determination policy is the mating restriction introduced into the crossover algorithm, to prevent creating lethals, because it is almost impossible to create competitive offspring if the parents cover different subsets of the training instances. However, it would be useful to study if there is any feasible way to recombine successfully individuals with different default class. If we achieve this objective, perhaps we can reduce the population size requirements of the *auto* policy.

Table 5.9: Results of the tests comparing the studied default class policies to the original configuration using pop. size 400

Domain	Result	Default rule policy				
		Disabled	Major	Minor	Auto	Major+Minor
bpa	Training acc.	79.3±1.7	82.0±1.4	80.7±1.4	81.0±1.6	82.0±1.4
	Test acc.	64.0±7.5	62.6±7.5	64.4±6.9	64.5±7.3	62.6±7.5
	#rules	6.8±1.0	8.9±1.4	8.3±1.6	8.7±1.4	8.9±1.4
bps	Training acc.	84.9±0.9	86.2±0.7	87.1±0.6	86.9±0.8	87.1±0.6
	Test acc.	80.4±4.5	80.9±3.8	81.6±3.8	81.2±3.9	81.6±3.8
	#rules	5.1±0.4	6.1±1.1	5.9±1.0	5.8±1.0	5.9±1.0
gls	Training acc.	80.8±2.5	83.8±1.6	81.3±2.1	79.5±1.7	83.8±1.6
	Test acc.	66.8±7.0	69.1±7.7	68.0±8.3	67.1±7.4	69.1±7.7
	#rules	6.5±0.7	6.8±0.8	7.5±0.9	6.7±0.8	6.8±0.8
h-s	Training acc.	90.1±1.0	92.0±0.9	92.4±0.8	92.2±0.8	92.4±0.8
	Test acc.	79.4±7.0	79.2±5.8	81.6±6.9	81.2±6.6	81.6±6.9
	#rules	6.6±0.8	7.8±1.3	7.4±1.2	7.4±1.2	7.4±1.2
ion	Training acc.	96.1±0.6	95.9±0.8	97.1±0.7	96.9±0.7	97.1±0.7
	Test acc.	93.5±3.5	90.4±4.3	93.4±3.5	92.8±4.0	93.4±3.5
	#rules	2.3±0.7	5.7±1.2	2.6±0.7	2.6±0.9	2.6±0.7
lrn	Training acc.	75.7±1.7	77.2±0.8	75.8±1.4	75.7±1.0	77.2±0.8
	Test acc.	68.0±5.0	69.1±5.4	68.7±5.2	69.1±4.9	69.1±5.4
	#rules	8.4±1.9	9.5±1.6	9.3±1.9	8.8±1.8	9.5±1.6
mmg	Training acc.	80.3±1.7	83.4±1.3	83.4±1.3	83.5±1.1	83.4±1.3
	Test acc.	65.9±8.3	69.0±8.0	67.3±8.9	69.7±7.7	69.0±8.0
	#rules	6.5±0.8	6.5±0.9	6.8±1.0	6.6±0.9	6.5±0.9
pim	Training acc.	80.0±1.0	81.5±0.7	81.2±0.7	81.4±0.7	81.5±0.7
	Test acc.	74.7±4.6	75.2±4.4	74.8±4.7	74.9±4.6	75.2±4.4
	#rules	5.3±0.6	6.3±1.1	5.8±0.9	6.1±1.0	6.3±1.1
son	Training acc.	92.7±1.5	96.7±1.1	95.3±1.3	96.1±1.3	96.7±1.1
	Test acc.	71.3±9.4	76.2±9.1	74.6±10.1	76.3±8.9	76.2±9.1
	#rules	6.7±1.0	7.6±1.3	7.7±1.5	7.6±1.4	7.6±1.3
thy	Training acc.	97.6±0.9	98.6±0.7	98.6±0.7	98.3±0.8	98.6±0.7
	Test acc.	91.5±6.2	92.0±5.2	92.4±4.8	91.4±5.6	92.4±4.8
	#rules	5.2±0.5	5.7±0.7	5.4±0.6	5.5±0.6	5.4±0.6
veh	Training acc.	71.9±1.9	74.1±1.3	74.2±1.2	72.6±1.3	74.2±1.2
	Test acc.	66.9±4.3	67.6±4.2	68.3±4.5	67.9±4.8	68.3±4.5
	#rules	6.5±1.3	9.4±1.8	10.0±1.8	8.4±1.8	10.0±1.8
wbcd	Training acc.	97.7±0.4	98.3±0.3	98.5±0.4	98.4±0.4	98.5±0.4
	Test acc.	95.7±2.3	95.0±2.6	95.7±1.9	95.8±1.9	95.7±1.9
	#rules	2.6±0.8	5.8±1.1	3.3±0.7	3.2±0.7	3.3±0.7
wdbc	Training acc.	97.2±0.8	98.0±0.5	97.9±0.6	97.8±0.6	98.0±0.5
	Test acc.	93.9±2.9	94.4±3.1	94.4±3.2	94.4±3.1	94.4±3.1
	#rules	4.3±1.2	4.8±1.1	4.2±0.7	4.5±0.9	4.8±1.1
wine	Training acc.	99.4±0.6	99.7±0.4	99.8±0.3	99.6±0.4	99.8±0.3
	Test acc.	94.1±6.0	93.2±6.4	92.0±6.5	93.2±6.3	92.0±6.5
	#rules	3.8±0.7	3.7±0.6	4.2±0.5	3.8±0.7	4.2±0.5
wpbc	Training acc.	84.9±2.8	89.9±1.8	87.1±3.3	89.0±2.1	89.9±1.8
	Test acc.	76.6±6.7	75.3±7.0	72.4±9.1	76.3±7.1	75.3±7.0
	#rules	2.8±0.9	3.9±0.9	4.4±1.2	3.7±1.0	3.9±0.9
ave	Training acc.	87.2±8.8	89.2±8.3	88.7±8.6	88.6±8.9	89.3±8.3
	Test acc.	78.8±11.5	79.3±10.7	79.3±11.1	79.7±10.8	79.7±11.7
	#rules	5.3±1.7	6.6±1.7	6.2±2.1	6.0±2.0	6.2±2.2

Table 5.10: Summary of the statistical t-tests applied to the results of the default rule experimentation with a population size of 400 and using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

Policy	Disabled	Major	Minor	Auto	Major+Minor	Total
Disabled	-	2	1	0	0	3
Major	1	-	1	0	0	2
Minor	1	3	-	0	0	4
Auto	1	3	1	-	0	5
Major+Minor	2	3	1	0	-	6
Total	5	11	4	0	0	

Another alternative is developing more sophisticated heuristics to combine the simple policies, although they have more computational cost, because the tests show that they cannot correctly choose the suitable policy in all datasets. More interesting would be to develop a method that would only need some short runs, instead of running a full test for each candidate policy. Of course, it is clear that this approach would require a very solid statistical validation in order to assure that the decision taken is correct.

5.8 Summary of the chapter

This chapter describes the research done on methods that extend the rule-based and decision-list-style knowledge representations for a Pittsburgh Learning Classifier System by using a static default rule. These kind of systems tend to generate an emergent default rule, which can increase the performance of the system. By forcing the representation of a default rule, we intended to guarantee these positive effects.

Simple policies such as using the majority/minority class as the default class perform quite well compared to the original system. However, they perform poorly on certain datasets showing a somewhat lack of robustness. We can integrate the best results of both policies by using the simple heuristic of selecting the policy with more training accuracy. This mechanism introduces a good performance boost, but doubles the run-time.

For this reason, we have developed a mechanism that decides automatically the class for the default rule. This technique works by integrating in a single population individuals using all possible default classes, and letting them compete among themselves. This approach has a problem, however, which is providing a fair competition framework, because each default rule

can have a different learning rate. In order to achieve this fairness, we use a niched tournament selection that guarantees that all niches (different default rules) survive in the population until they can compete successfully by themselves. This automatic mechanism performs best when we increase the population size, which is an usual requirement in most systems that use niching, because we have to guarantee that each niche has enough individuals to ensure building block supply and thus successful and reliable learning.

The increase in population size for the majority/minority policies, however, showed no performance increase and even some performance decrease, suggesting the amplification of the policy weaknesses These weaknesses are derived from over-learning, which is reflected in the larger training accuracy and larger average rule set sizes and also on the statistical tests.

Although the automatic policy does not outperform the major+minor policy, the accuracy difference is quite small in most datasets and the computational cost is significantly lower. Therefore, it appears that in most situations the automatic policy is the best method.

Chapter 6

The adaptive discretization intervals rule representation

This chapter describes the contributions of this thesis to the area of representations for real-valued attributes, proposing a representation called adaptive discretization intervals rule representation*. The approach chosen to handle these attributes is by using a discretization process, but in a special way: the intervals used in the rules are created by joining together some adjacent cut-points proposed by a discretization algorithm. Also several discretization algorithms are used at the same time, letting the system choose the most suitable one for each dataset. With these two characteristics, the proposed representation gains robustness and has an efficient exploration of the search space.*

The chapter is structured as follows: section 6.1 will contain a larger introduction to the chapter explaining the motivations of the main characteristics of ADI*. Next, section 6.2 will show some related work, followed by section 6.3 with an extensive description of the basic mechanisms of the representation. Section 6.4 will illustrate the behavior of the basic representation on some datasets, which will lead to the identification of some problems, that will be corrected with the new operator described in section 6.5. Section 6.6 will show the extensive tests done to the representation, first with several well-known discretization algorithms alone,then combining them with some criteria and finally comparing the* ADI *representation with two alternative real-valued representations. Finally, section 6.7 will show some discussion and further work and section 6.8 will provide a summary of the chapter.*

6.1 Introduction and motivation

There are two major options when with real-valued attributes, seen from a very simplistic point of view: handle them as nominal attributes are handled or handle the real values directly. In order to perform the former, a discretization process is needed to convert the real values into a finite set of intervals that are considered as nominal values.

The discretization methods (section 2.5 contents a summary of several discretization techniques), however, brings up some problematic issues. The first is that there is no discretization algorithm that allows the learning system to perform well in all datasets. This is a natural fact because discretizers, like learning algorithms, also introduce inductive bias into the solutions generated and, therefore, are affected by the *selective superiority problem* (Brodley, 1993). If a discretization algorithm with less bias (like the unsupervised uniform-width and uniform-frequency ones) is chosen there is another problem: probably there will be several irrelevant cut points, which will produce a search space bigger than necessary, wasting computation time.

This chapter describes the contributions done in discretization-based representations for real-valued attributes that can handle the two previous issues: the **adaptive discretization intervals rule representation**. This representation constructs intervals using as "low-level bricks" the cut points proposed by a discretization algorithm. The intervals constructed can be merged with adjacent intervals or split (having a minimum size: the low-level intervals). In this way, the problem of useless search space is solved because the representation collapses the search space where it is possible. Also, several discretization algorithms can be used at the same time, allowing the system to choose for each domain (and even for each attribute) the most suitable discretization algorithm, addressing the other issue described above.

6.2 Related work

Discretization is not the only way to handle real-valued attributes in Evolutionary Computation based Machine Learning systems. Some examples are induction of decision trees (either axis-parallel or oblique), by either generating a full tree by means of genetic programming operators (Llorà & Garrell, 2001b) or using an heuristic method to generate the tree and using a Genetic Algorithm and an Evolution Strategy to optimize the test performed at each node (Cantu-Paz & Kamath, 2003). Other examples are inducing rules with real-valued intervals (Wilson, 1999; Stone & Bull, 2003) or generating an instance set used as the core of a *k-NN* classifier (Llorà & Garrell, 2001b). A broad range of systems perform classifications tasks using

evolutionary knowledge representations based on fuzzy logic. A good review of these systems is (Cordón, Herrera, Hoffmann, & Magdalena, 2001).

Also, there are other systems, like *ADI*, that perform evolutionary induction of rules based on discretization (Giráldez, Aguilar-Ruiz, & Riquelme, 2003; Divina, Keijzer, & Marchiori, 2003). A comparison of *ADI* with these two methods is found in (Aguilar, Bacardit, & Divina, 2004). Other approaches to concept learning, e.g., Neural Network, do not need any discretization for handling numerical values.

If our *ADI* method is able to find the correct cut points, the performance of the system should be quite competitive if compared to all the axis-parallel methods described above.

6.3 Basic mechanisms of the ADI representation

The semantic structure of each rule in ADI is taken from **GABIL** (DeJong & Spears, 1991): Each rule consists of a condition part and a classification part: $condition \rightarrow decision$. Each condition is a Conjunctive Normal Form (CNF) predicate defined as:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

Where A_i is the *i*th attribute of the problem and V_i^j is the *j*th value that can take the *i*th attribute. This kind of predicate can be encoded into a binary string in the following way: if we have a problem with two attributes whose values can be {1,2,3}, a rule of the form "If the first attribute has value 1 or 2 and the second one has value 3 then we assign class 1" will be represented by the string $110|001|1$.

In *GABIL* for each attribute we would use a set of static discretization intervals instead of nominal values. The intervals of the ADI representation are not static, but they evolve through the iterations splitting and merging among them (having a minimum size called *micro-interval*). Thus, the binary coding of the *GABIL* representation is extended as represented in figure 6.1, also showing the split and merge operations.

In order to make the interval splitting and merging part of the evolutionary process, we have to include it in the GAs genetic operators. We have chosen to add to the GA cycle two special stages applied to the offspring population after the mutation stage. The new GA cycle is represented in figure 6.2. For each stage (split and merge) we have a probability (p_{split} or p_{merge}) of applying the operation to an attribute term. Figure 6.3 contains the code that deals with this probability for the *merge* operator. The code for the split operator is similar.

A complete specification of the ADI representation is shown as follows:

Figure 6.1: Adaptive intervals representation and the split and merge operators.

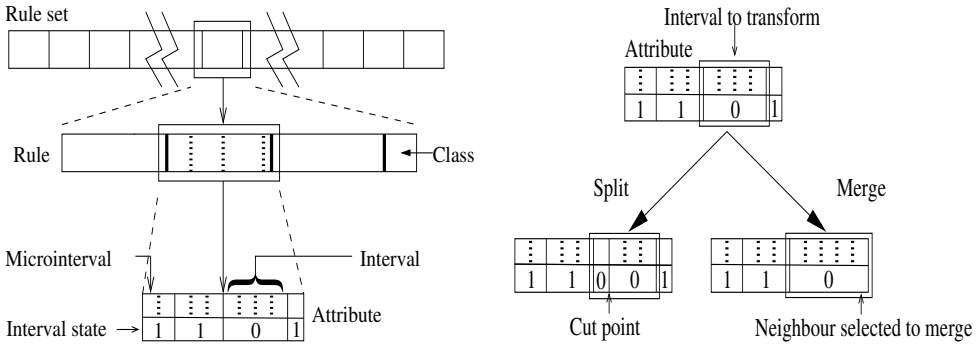


Figure 6.2: Extended GA cycle for ADI representation with split and merge stages

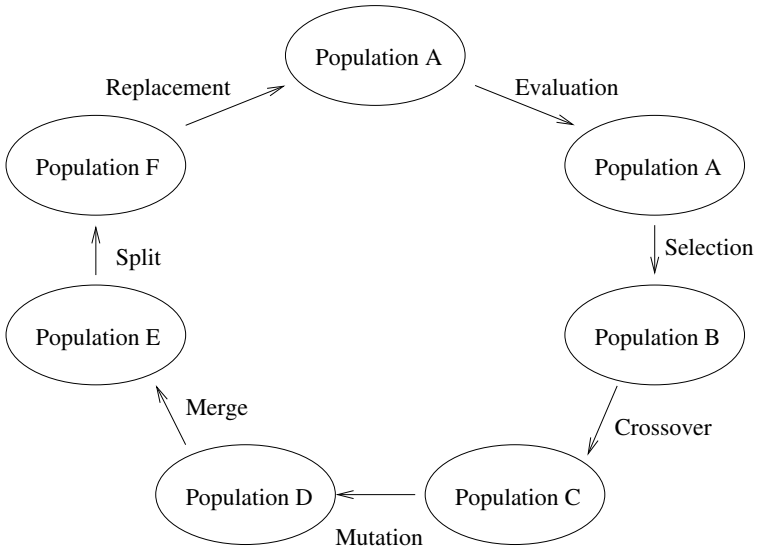


Figure 6.3: Code of the application of the merge operator in ADI

```
ForEach Individual i of Population
        ForEach Rule j of Population individual i
                ForEach Attribute k of Rule j of Population individual i
                        If random [0..1] number < p_merge
                           Select one random interval of attribute term k
                                   of rule j of individual i
                           Apply a merge operation to this interval
                        EndIf
                EndForEach
        EndForEach
EndForEach
```

1. A set of static discretization intervals (called *micro-intervals*) is assigned to each attribute term of each rule of each individual.
2. The intervals of the rule are built joining together adjacent *micro-intervals*.
3. Attributes with different number and sizes of *micro-intervals* can coexist in the population. The evolution will choose the correct discretization for each attribute.
4. For computational cost reasons, we will have an upper limit in the number of intervals allowed for an attribute, which in most cases will be less than the number of *micro-intervals* assigned to each attribute.
5. The mutation operator will affect the semantic part of the rule: the states (0 or 1) of each interval in the rule. In this way, the mutation operator is identical to the one used in *GABIL*.
6. When we split an interval, we select a random point in its *micro-intervals* to break it.
7. When we merge two intervals, the state (1 or 0) of the resulting interval is taken from the one which has more *micro-intervals*. If both have the same number of *micro-intervals*, the value is chosen randomly.
8. The discretization assigned in the initialization stage to each attribute term is chosen from a predefined set.
9. The number and size of the initial intervals is selected randomly.
10. The cut points of the crossover operator can only take place in attribute terms boundaries, not between intervals. This restriction takes place in order to maintain the semantic correctness of the rules.

Figure 6.4: Code of the split operator in ADI

Split operator
Input : $Attribute$ to split
If number of intervals in $Attribute = maxIntervals$ (user defined)
  Do nothing
Else
  $interval =$ choose randomly an interval from $Attribute$
  If number of *micro-intervals* in $interval > 1$
    $cutPoint =$ random cut-point between the *micro-intervals* of $interval$
    $int_1, int_2 =$ Split $interval$ in two by $cutPoint$
    truth value of $int_1 =$ truth value of $interval$
    truth value of $int_2 =$ truth value of $interval$
    Remove $interval$ from $Attribute$
    Add int_1 and int_2 to $Attribute$
    Increase interval count of $Attribute$
  EndIf
EndIf
Output : $Attribute$

11. The bloat control methods used (like the *hierarchical selection* or the MDL-based fitness function, described in chapter 8) promote individuals with both reduced number of rules and also reduced number of intervals. If only rules were used, the search space collapsing effect of the merge operator would not be effective.

Figures 6.4,6.5 and 6.6 show the code for the split, merge and attribute initialization operators, respectively. Figure 6.7 shows the matching process for an rule in *ADI*.

The *ADI* knowledge representation has changed over the time. The first version (Bacardit & Garrell, 2002a) only used one discretizer at the same time (and only uniform-width discretizers), and had the merge and split operators integrated into the mutation operator. The next release (Bacardit & Garrell, 2002b) already used several uniform-width discretizers at the same time and an individual-wise probability of split and merge. Finally, the probabilities of split and merge were changed to attribute-wise ones (Bacardit & Garrell, 2003c) and uniform and non-uniform discretization algorithms were integrated into the representation.

Figure 6.5: Code of the merge operator in ADI

Merge operator
Input : $Attribute$ to merge
If number of intervals in $Attribute > 1$
  $interval =$ choose randomly an interval from $Attribute$
  $neighbour =$ choose randomly an adjacent interval to $interval$
  If number of *micro-intervals* in $interval >$ number of *micro-intervals* in $neighbour$
    $newValue =$ truth value of $interval$
  Else If number of *micro-intervals* in $interval <$ number of *micro-intervals* in $neighbour$
    $newValue =$ truth value of $neighbour$
  Else
    If random$[0, 1] < 0.5$
      $newValue =$ truth value of $interval$
    Else
      $newValue =$ truth value of $neighbour$
    EndIf
  EndIf
  $newInterval =$ merge $interval$ and $neighbour$
  Truth value of $newInterval = newValue$
    Remove $interval$ and $neighbour$ from $Attribute$
    Add $newInterval$ to $Attribute$
    Decrease interval count of $Attribute$
EndIf
Output : $Attribute$

Figure 6.6: Code of the attribute initialization in ADI

Initialization operator
Input : nothing
$discretizer$ = choose randomly a discretization algorithm from our predefined pool
$microIntervals$ = number of cut points of $discretizer + 1$
$maxAllowedIntervals$ = min($globalMaxIntervals$,$microIntervals$)
If $maxAllowedIntervals > 2$
  $numIntervals$ = random$[2..maxAllowedIntervals]$
Else
  $numIntervals = maxAllowedIntervals$
EndIf
Discretizer of $Attribute = discretizer$
Interval count of $Attribute = numIntervals$
For $i = 0$ to $numIntervals - 1$
  If $i < numIntervals - 1$
    $microInt$ = random$[1..(microIntervals - (numIntervals - i - 1))]$
  Else
    $microInt = microIntervals$
  EndIf
  $microIntervals = microIntervals - microInt$
  $interval$ = Create an interval with size $microInt$
  If random$[0, 1] < probabilityOfTrue$ (user defined)
    Truth value of $interval = true$
  Else
    Truth value of $interval = false$
  EndIf
  Insert $interval$ in $Attribute$
EndFor
Output : $Attribute$

Figure 6.7: Code of the matching process in ADI

Matching process
Input : $Rule$,$Instance$
// *To simplify, we suppose that all attributes are real-valued*
$matchOK = true$
ForEach $Attr$ in $Instance$ **While** $matchOK = true$
    $Attribute =$ Attribute $Attr$ of $Rule$
    $disc =$ Discretizer assigned to $Attribute$
    $interval =$ Value assigned to $Attr$ by discretizer $disc$
    $sumMicroInt = 0$
    $found = false$
    ForEach $Interval$ in $Attribute$ **While** $found = false$
        $sumMicroInt = sumMicroInt +$ number of *micro-intervals* in $Interval$
        If $interval < sumMicroInt$
            $found = true$
            If truth value of $Interval = false$
                $matchOK = false$
            EndIf
        EndIf
    EndForEach
EndForEach
Output : $matchOK$

6.4 Behaviour of the basic ADI knowledge representation

In this section the behavior of the ADI representation presented in previous section is analyzed. This short analysis leads us to identify a flaw in the representation, which will be fixed with the operator presented in next section.

6.4.1 Discretizers in the population. Finding the ideal discretizer

The first issue we want to examine is the distribution of discretizers in the population. In the ADI representation, the initialization stage assigns a random discretizer from our predefined pool to each attribute term of each rule of each individual. Through the iterations, the discretizers of the best individuals survive, but no new discretizers are inserted into the population. This brings up the question of how the number of attributes in the population assigned to each discretizer evolves through the iterations. This information was extracted from the population and it is represented in figure 6.8. This figure shows the evolution of the discretizer proportions for 4 problems (*bre,irs,mmg,pim*). The configuration of these tests and the ones in next section is summarized in table 6.1. The discretizers chosen for these tests are the used in previous work (Bacardit & Garrell, 2003c): uniform-width discretizer of 4,5,6,7,8,10,15,20,25 intervals. We show this figure to compare the behavior among the datasets. Therefore, the same Y scale is used in all plots.

Figure 6.8 shows that all discretizers start the evolutionary process with a proportion approximately of 1/number of discretizers. Later on, the proportions change through the iterations. We can see that the proportions for all the datasets do not diverge too much from their initial value, with the exception of the *iris* dataset. This behavior made us wonder if the system had managed to identify the ideal discretizer for this dataset. Thus we repeated the tests for these four datasets but using only the discretizer most frequent for each problem. The results are detailed in table 6.2.

We can see that the only dataset where there is accuracy increase when we are using only one discretizer is *iris*. It would be interesting to determine if there are other datasets where the evolution of the discretizer proportions presents the same behavior, and check if they manage also to identify the ideal discretizer. Unfortunately, we could not find any more dataset presenting this behavior.

Table 6.1: Settings of GAssist for the ADI behavior tests

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	300
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	disabled
Number of strata of ILAS windowing	2
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of active rules + 3
Hierarchical selection operator	
Iteration of activation	25
Threshold	0.01

Table 6.2: Results of the experiment of using only the discretizer with more proportion in the population

Dataset	Original accuracy	Accuracy with one discretizer	Discretizer
bre	95.6±2.2	95.5±1.8	4 intervals
irs	95.9±3.9	97.8±3.1	6 intervals
mmg	65.0±9.0	63.1±8.8	25 intervals
pim	74.4±4.7	74.2±3.7	4 intervals

Figure 6.8: Evolution of the discretizer proportions in the population for the *bre*,*iris*,*mmg*,*pim* datasets

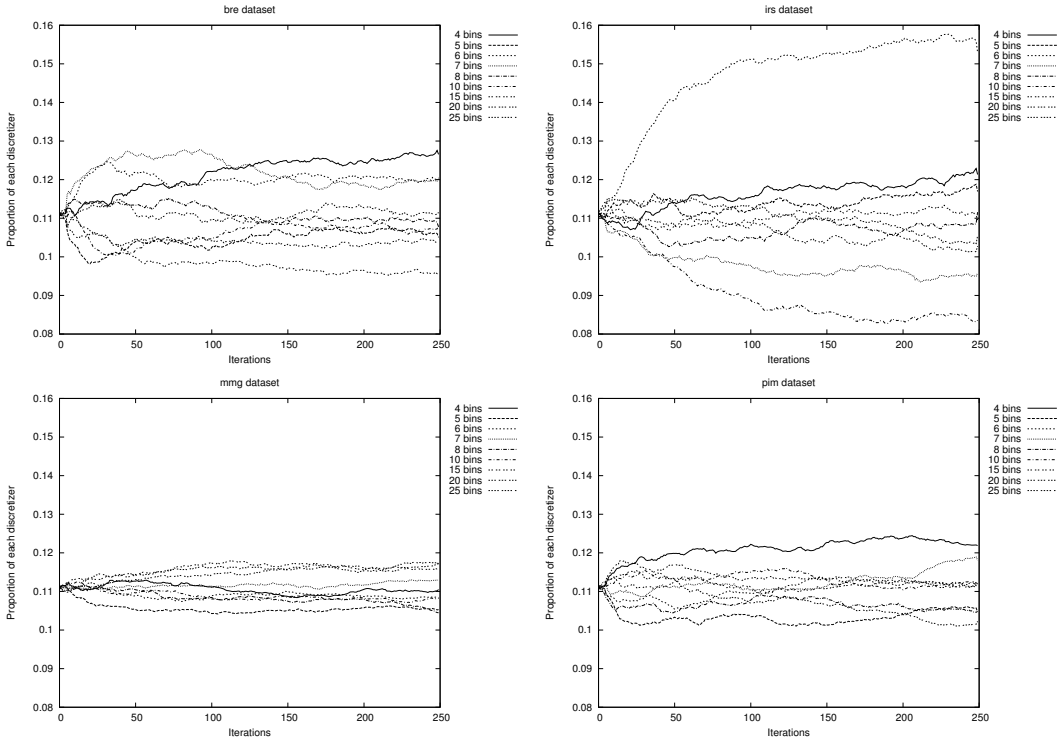
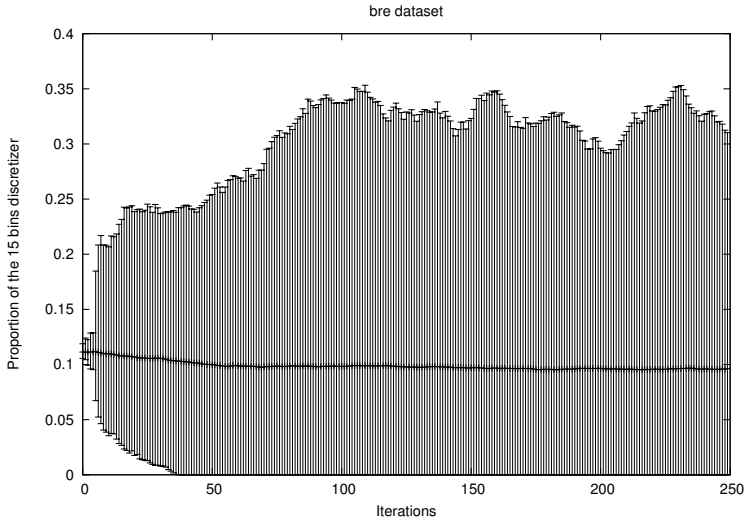


Figure 6.9: Evolution of the proportions of the uniform-width discretizer with 15 intervals in the population for the bre dataset



6.4.2 Discretizers in the population. Survival of the discretizers

The next issue to analyze of the *ADI* representation is also related to the discretizer proportions in the population. In figure 6.9 we show the evolution of the average proportion of the 15 intervals discretizer for the *bre* dataset, but this time using error bars. We can extract an important piece of evidence: In some runs, this discretizer disappeared from the population in less than 30 iterations. Other discretizers and datasets show the same behavior. Is this effect good or bad? Ideally the *GA* should choose the ideal discretizer for each domain and attribute. However, in most situations the system is not prepared to choose correctly in few iterations because it has not learned enough. It is clear that, in order to avoid this situation, we have to create some kind of mechanism that is able to introduce new discretizers into the population through the evolutionary process, which is presented in next section.

6.5 The reinitialize operator

This section presents the work developed (Bacardit & Garrell, 2004) to fix the problem of good discretizers disappearing too soon from the population. A mechanism that allows all discretizers to survive in the population until the system can choose correctly among them is

Figure 6.10: Steps of the reinitialize operator

1. We have an attribute term of a rule in the population selected for reinitialize
2. We select randomly a discretizer from our predefined pool
3. We assign a number of intervals to the attribute. This number is randomly chosen between 2 and min(number of *micro-intervals*, maximum allowed intervals per attribute)
4. We assign randomly a number of consecutive *micro-intervals* to each interval of the attribute. Every interval must have at least one *micro-interval*
5. We assign a random truth value (0 or 1) to each interval

needed.

What form should such survival mechanism take? Probably the most suitable form would be an operator that changes the discretizer used by an attribute but maintaining, as much as possible, the semantic structure of the attribute. That is, finding a set of intervals built over the new discretizer as close as possible to the old ones. Unfortunately, an operator like this can present a huge computational cost considering that it can be common to deal with datasets that have hundreds of cut points. Therefore, we start by studying a more simple operator, called **reinitialize**. This operator repeats the process done in the initialization stage of the *GA*, but only for the selected individual and attribute term, as represented in figure 6.10.

This operator is applied after the merge and split stages, and the probability controlling it is also defined for each attribute-term. In order to assign a good value to this probability we did some tests with some probability values (0.0025,0.005,0.01,0.015). We reproduce only the results for the *mmg* and *pim* datasets because they illustrate two different kinds of behavior. The results are in table 6.3, where we can see a correlation between the probability increase and the a decrease of obtained training accuracy and more rules and intervals per attribute. However, test accuracy does not show these trends. While the *pim* dataset does not benefit from this operator, the *mmg* dataset has a notable accuracy increase, considering that we are comparing two versions of the same system.

Therefore, we can see that the operator is beneficial in some domains but its effects are too much aggressive (creating poor solutions) when applied to other datasets. The reinitialize operator needs to be redefined to have a milder behavior. The simplest way to achieve this goal is to redefine the probability controlling the operator. The new probability decreases linearly through the iterations until it achieves value 0 at last iteration. This fix allows the system to explore more aggressively in the early iterations and later on, in the final iterations, refine the good solutions. We repeated the short test with the same datasets, using as initial probabilities the values 0.01,0.02,0.03,0.04. The results are in table 6.4.

There are some interesting differences from the previous results. The training accuracy of

Table 6.3: Short tests of the reinitialize operator

Dataset	Reinit. prob.	Training acc.	Test acc.	# of rules	Interv. per attr
mmg	0.0000	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.0025	78.4±1.8	65.7±8.8	6.5±1.0	2.5±0.1
	0.0050	78.2±1.7	66.8±8.8	6.5±0.8	2.5±0.1
	0.0100	77.5±1.7	67.3±9.5	6.5±0.9	2.6±0.1
	0.0150	76.4±1.8	67.5±9.1	6.6±1.0	2.7±0.1
pim	0.0000	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.0025	78.1±0.9	74.4±4.5	5.1±0.6	2.2±0.1
	0.0050	78.1±1.0	74.4±4.7	5.3±0.7	2.3±0.1
	0.0100	77.9±1.0	74.3±4.3	5.2±0.8	2.3±0.1
	0.0150	77.7±1.0	74.1±5.1	5.1±0.6	2.4±0.1

Table 6.4: Short tests of the improved reinitialize operator

Dataset	Initial reinit. prob.	Training acc.	Test acc.	# of rules	Interv. per attr
mmg	0.00	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.01	78.8±1.6	65.7±9.4	6.5±0.8	2.4±0.1
	0.02	78.4±1.7	66.2±9.4	6.5±1.0	2.5±0.1
	0.03	78.4±1.5	67.1±8.1	6.4±0.7	2.5±0.1
	0.04	78.0±1.8	67.2±8.0	6.6±1.0	2.5±0.1
pim	0.00	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.01	78.7±1.0	74.6±4.5	5.3±0.9	2.3±0.1
	0.02	78.7±1.0	74.6±4.4	5.4±1.0	2.3±0.1
	0.03	78.6±1.0	75.1±4.5	5.3±0.7	2.3±0.1
	0.04	78.5±1.1	74.3±4.7	5.2±0.7	2.3±0.1

tests with the reinitialize operator is slightly higher that the original configuration. This shows that we have achieved the objective of creating an operator that helps in exploring the search space for better solutions while being soft enough so as not to destroy these solutions in the final iterations. Even more interesting is the test accuracy, because now we obtain an accuracy increase over the original *ADI* configuration in both domains. Extensive experimentation of this reinitialize operator was conducted (Bacardit & Garrell, 2004), showing that the most suitable initial probability of reinitialization was 0.02. New tests of this kind will be repeated in next section, experimenting this time with all kinds of discretization algorithms.

6.6 Which are the most suitable discretizers for ADI?

This section will describe the extensive experimentation performed to determine which is the most suitable set of discretization algorithms for the *ADI* knowledge representation. First, some tests will be conducted using each studied discretization algorithm alone. The performance of the discretizers by itself will be used by several criteria to propose some combination of discretization algorithms, that will be tested will different settings of the reinitialize operator, in order to find the best configuration for *ADI*. Finally, this best configuration will be compared to two alternative knowledge representations handling directly real values.

6.6.1 Testing each discretization algorithm alone

The aim of this first set of experiments is to determine which are the candidate discretization algorithms that will be used together in later experiments. The chosen discretization algorithms (described in section 2.5) are the following:

- Uniform-width (Liu, Hussain, Tam, & Dash, 2002)
- Uniform-frequency (Liu, Hussain, Tam, & Dash, 2002)
- Id3 (Quinlan, 1986)
- Fayyad & Irani (Fayyad & Irani, 1993)
- Màntaras (Cerquides & de Mantaras, 1997)
- USD (Giráldez, Aguilar-Ruiz, Riquelme, Ferrer, & Rodríguez, 2002)
- ChiMerge (Kerber, 1992)

Table 6.5: Discretizers used in the ADI experimentation with the chosen sets of parameters

Discretization algorithm	parametrizations
Uniform-width	5,10,15,20,25 bins
Uniform-frequency	5,10,15,20,25 bins
ID3	—
Fayyad & Irani	—
Màntaras	—
USD	—
ChiMerge	significance levels: 0.01,0.5
Random	—

Also, as a baseline, a random discretization algorithm will be used. This discretizer chooses as cut points a random subset of the mid-points between all values in the attribute domain. Some of the chosen discretization algorithms need to be parametrized. The chosen parametrizations of all discretizers are described in table 6.5. The total number of discretizers tested (counting the different parametrizations of each discretizer) will be 17. Table 6.6 shows the configuration used in the test, tables A.1 through A.14 of appendix A show the results of these tests for each tested dataset, and table 6.7 shows the average results over all datasets.

From the averages of the results we can see some interesting facts. There are some differences between the training accuracy of some methods, especially between ID3 and Màntaras. This is a consequence of the number of cut-points proposed by each discretizer (table 6.8 shows the average number of cut points per attribute for each discretization algorithm). ID3 is the discretizer generating more intervals (leaving out Random), and Màntaras the one generating less intervals. However, this difference does not translate into test accuracy, where the difference is very small. The number of rules and the run time of the Màntaras discretizer also reflect that this discretizer generates a small number of cut points.

If we look in general at the most important result, the test accuracy, we see that on average the accuracy differences between all methods are small, even for the random discretizer. However, we cannot say that all discretizers all equally good, because if we look at the results of any individual dataset we can see large accuracy differences. Of course, this was expected, it only reflects the specific inductive bias of each discretization algorithm. To gain more insight on the discretizers performance we need to apply statistical tests to the results. Table 6.9 summarizes the pairwise t-tests with Bonferroni correction applied to the results. For each discretizer it shows how many times it has been able to outperform significantly another discretizer (with a 95% confidence level) and also how many times it has been outperformed.

Table 6.6: Settings of GAssist for the ADI single discretizers tests

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	400
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	auto
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Maximum number of intervals	5
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of active rules + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

Table 6.7: Averages of the results of the ADI tests with a single discretizer

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	90.3±8.3	80.3±11.9	5.8±2.0	46.9±32.6
ChiMerge 0.50	91.3±7.6	80.7±11.8	6.3±2.4	48.6±33.0
Uniform frequency 10	90.5±7.8	81.1±11.5	6.8±2.4	47.1±31.1
Uniform frequency 15	90.6±7.7	80.8±11.7	6.8±2.4	47.4±31.5
Uniform frequency 20	90.7±7.6	81.0±11.4	6.8±2.4	47.8±31.8
Uniform frequency 25	90.7±7.5	80.9±11.4	6.7±2.5	47.9±32.4
Uniform frequency 5	89.2±8.4	80.9±11.3	6.6±2.2	46.2±31.0
ID3	91.3±7.4	81.1±11.5	6.6±2.4	46.5±30.7
Màntaras	85.0±12.8	80.9±12.6	5.1±1.9	37.5±28.9
Fayyad	86.4±11.4	80.7±12.3	4.9±1.8	41.0±30.7
Random	89.1±8.9	80.0±11.9	6.3±2.0	46.7±31.8
Uniform width 10	89.2±8.8	80.7±12.4	6.5±2.5	45.0±30.7
Uniform width 15	89.8±8.4	81.0±12.2	6.6±2.6	45.6±30.7
Uniform width 20	90.1±7.9	81.1±11.6	6.6±2.5	45.4±30.5
Uniform width 25	90.1±8.1	80.8±12.0	6.6±2.6	45.5±30.5
Uniform width 5	87.1±10.6	80.0±13.6	6.1±2.1	43.2±28.9
USD	90.2±8.3	80.3±12.0	6.2±2.1	44.4±29.9

Table 6.8: Average number of cut-points per attribute for the tested discretization algorithms

Discretizer	bal	bpa	gls	h-s	ion	irs	lrn	mmg	pim	thy	wbcd	wdbc	wine	wpbc	Ave.
ChiMerge 0.01	1.61	4.48	5.20	2.15	8.60	4.00	5.57	5.80	5.49	3.56	2.73	9.03	3.90	4.70	4.77
ChiMerge 0.50	3.00	9.56	9.32	4.35	9.41	7.10	10.00	10.00	9.74	10.00	6.34	10.00	10.00	10.00	8.49
Uniform frequency 10	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00
Uniform frequency 15	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00
Uniform frequency 20	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00
Uniform frequency 25	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00
Uniform frequency 5	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
ID3	4.00	43.06	71.09	21.24	61.54	13.60	109.67	81.15	100.75	27.18	8.74	151.05	50.79	61.11	57.50
Màntaras	2.38	0.02	0.74	0.74	1.34	2.62	3.02	0.70	1.12	3.71	2.67	1.83	1.88	0.03	1.63
Fayyad	2.00	0.33	1.52	0.75	3.10	2.12	3.17	0.98	1.45	2.63	2.96	1.98	1.85	0.06	1.78
Random	1.47	23.86	51.18	13.60	106.18	13.82	67.96	96.95	72.24	31.87	4.53	234.41	44.55	78.37	60.07
Uniform width 10	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00
Uniform width 15	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00
Uniform width 20	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00	19.00
Uniform width 25	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00	24.00
Uniform width 5	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
USD	1.00	19.43	50.12	9.56	60.17	5.26	73.38	80.26	48.98	11.01	2.31	129.18	32.39	55.80	41.35

Table 6.9: Pairwise t-tests applied to the results of the tests with single discretizers

Discretizer	#times outperforming	#times outperformed
ChiMerge 0.01	10	36
ChiMerge 0.50	13	18
Uniform frequency 10	21	5
Uniform frequency 15	15	14
Uniform frequency 20	18	12
Uniform frequency 25	15	12
Uniform frequency 5	16	27
ID3	18	7
Màntaras	41	26
Fayyad	33	34
Random	11	37
Uniform width 10	25	24
Uniform width 15	23	15
Uniform width 20	31	5
Uniform width 25	20	10
Uniform width 5	25	45
USD	14	22

From the results of the t-tests we can see that the discretizer that is able to outperform significantly more times than anyone the other methods, the Màntaras discretizer, is also one of the most outperformed ones, showing that it lacks some robustness capacity, which probably is a consequence of generating so few cut-points. In some domains the loss of information created by the discretization becomes critical. We can see that the most robust discretizers (the ones being outperformed less times than anyone) are *Uniform-width 20*, *Uniform-frequency 10* and *ID3*. *ID3* having the best training accuracy average and showing also its robustness we can say that it is the best single discretization algorithm.

6.6.2 Testing the groups of discretization algorithms

In order to propose the groups of discretizers that are going to be tested we rely on the following criteria, that created the sets of discretizers described in table 6.10 :

1. The 8 discretizers that were significantly outperforming another discretization algorithm more times, based on the t-tests

2. The 4 discretizers that were significantly outperforming another discretization algorithm

Table 6.10: Selected sets of discretizers for the multiple discretizers ADI experimentation

Group	Discretizers in group
Group 1	Màntaras
	Fayyad & Irani
	Uniform-width 20
	Uniform-width 5
	Uniform-width 10
	Uniform-width 15
	Uniform-frequency 10
	Uniform-width 25
Group 2	Màntaras
	Fayyad & Irani
	Uniform-width 20
	Uniform-width 5
Group 3	Uniform-frequency 10
	Uniform-width 20
	ID3
	Uniform-width 25
	Uniform-frequency 20
	Uniform-frequency 25
	Uniform-frequency 15
	Uniform-width 15
Group 4	Uniform-frequency 10
	Uniform-width 20
	ID3
	Uniform-width 25
Group 5	Màntaras
	Fayyad & Irani
	USD
	ID3
Group 6	Uniform-width 4
	Uniform-width 5
	Uniform-width 6
	Uniform-width 7
	Uniform-width 8
	Uniform-width 10
	Uniform-width 15
	Uniform-width 20
	Uniform-width 25
Group 7	Uniform-frequency 4
	Uniform-frequency 5
	Uniform-frequency 6
	Uniform-frequency 7
	Uniform-frequency 8
	Uniform-frequency 10
	Uniform-frequency 15
	Uniform-frequency 20
	Uniform-frequency 25

more times, based on the t-tests (a subset of the previous discretizer)

3. The 8 discretizers that were significantly outperformed by another discretization algorithm less times, based on the t-tests

4. The 4 discretizers that were significantly outperformed by another discretization algorithm less times, based on the t-tests (a subset of the previous discretizer)

5. The discretization algorithms with no parameters

6. The set of uniform-width discretizers used in previous work (Bacardit & Garrell, 2004)

7. An equivalent set of discretizers to the previous one but using uniform-frequency discretization

Table 6.11: Averages of the results of the ADI tests with the groups of discretizers, without reinitialize

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	89.6±8.3	81.2±13.3	6.0±2.2	44.4±29.7
Group 2	88.9±8.9	81.3±13.3	5.7±2.0	42.6±28.8
Group 3	90.5±7.7	81.0±13.1	6.4±2.4	46.8±30.9
Group 4	90.5±7.8	81.1±13.2	6.4±2.4	45.9±30.3
Group 5	89.9±8.1	81.3±13.3	5.8±1.9	44.3±29.8
Group 6	89.3±8.5	80.8±13.7	6.2±2.2	44.7±30.0
Group 7	90.2±7.8	81.3±13.1	6.4±2.2	46.8±30.6

Testing the groups of discretizers without the reinitialize operator

The first experiments done on these groups of discretizers consists of an equivalent test to the previous tests using only a single discretizer, that is, without the reinitialize operator. The average results over all datasets of these tests appears in table 6.11. The full results of these tests are in appendix A. We can see from the averages of the test accuracy that the performance of the groups of discretizers, even if there is no increase, are more robust than using a single discretization algorithms. The range of average accuracies obtained goes from 80.8% to 81.3%, compared to the 80.0%-81.1% of the single discretizers, validating that the approach of mixing several discretizers is feasible.

Moreover, we compared statistically the performance of the best single discretizer (ID3) with these groups, using the usual pairwise t-tests. Table 6.12 shows the results of these tests, indicating by the few significant differences found that all configurations perform similarly. Maybe we can only discard group 6, because it is the one showing less robustness. What these results show is that the groups of discretization algorithms are unable to discover in all tests the most suitable discretizer. This picture changes slightly in the next set of tests, when we activate the reinitialization operator.

Testing the groups of discretizers with the reinitialize operator

The next set of tests will use the reinitialization operator. As we have described in section 6.5, previous tests (Bacardit & Garrell, 2004) showed that the most robust reinitialize probability (in the tested datasets) was 0.02. These previous tests only used group 6 of discretization algorithms. Also, some of the techniques used in the current tests (like the default rule mechanism or the MDL-based fitness function) were not used before. Therefore we will run the tests again to determine the best reinitialize probability for each group of discretizers.

Table 6.12: Results of the t-tests comparing the best single discretizer with the seven tested groups of discretizers without reinitialize operator, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	ID3	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Total
ID3	-	0	1	0	0	0	1	0	2
Group 1	0	-	0	0	0	0	0	0	0
Group 2	0	0	-	1	0	0	1	0	2
Group 3	0	0	1	-	0	0	0	0	1
Group 4	1	0	0	0	-	0	1	1	3
Group 5	0	0	0	0	0	-	1	0	1
Group 6	0	0	0	0	0	0	-	0	0
Group 7	0	0	1	1	0	0	1	-	3
Total	1	0	3	2	0	0	5	1	

We tested probabilities are 0.01,0.02,0.03,0.04, and table 6.13 shows the average results on all datasets. As usual, the full details of the results are in appendix A.

We can see that the reinitialize operator, using almost any of the tested probabilities, produces an accuracy increase. The ranges of average accuracies obtained are (81.2%-81.5%), (81.3%-81.6%), (81.3%-81.9%) and (81.4%-81.7%). These ranges are totally non-overlapped compared to the performance of the single discretizer tests, and almost non-overlapped compared to the tests without reinitialize. With these observations we can almost assure the benefits of the reinitialize operator, backing our previous results. The accuracy differences might seem small, but we want to remind that these results are averages over all discretizers, and we are only comparing different settings of the same system. Therefore, we think that these results are quite relevant.

The next goal is to determine, from these results, the best settings for each group of discretization algorithms. For this task we will rely, as usual, on the results of the statistical t-tests. This time we will compare, for each group of discretization algorithms, all experimented settings with and without reinitialize. The results of these statistical t-tests are shown in table 6.14.

Some different behavior can be observed from the results: group 1 is totally insensitive to the reinitialize operator, no significant differences were observed. On the other hand, group 6 benefits totally from reinitialize because the only outperformed configuration was the one without the operator. This result was expected, as this operator was designed using only this group of discretizers. However, groups 2, 3 and 4 to a lesser degree also benefit from reinitialize. The opposite case is group 5, the one with only supervised and parameter-less discretizers,

Table 6.13: Averages of the results of the ADI tests with the groups of discretizers with reinitialize

Reinit. prob.	Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
0.01	Group 1	90.1±8.0	81.5±12.9	5.9±2.2	45.1±30.6
	Group 2	89.5±8.5	81.5±13.0	5.6±2.1	43.3±29.8
	Group 3	90.8±7.6	81.4±13.2	6.3±2.5	47.2±31.7
	Group 4	90.8±7.6	81.2±13.3	6.3±2.4	45.9±30.8
	Group 5	90.4±7.8	81.3±13.2	5.7±2.1	44.4±30.4
	Group 6	89.7±8.2	81.2±13.5	6.0±2.3	45.3±30.7
	Group 7	90.4±7.8	81.2±13.1	6.3±2.3	47.1±31.5
0.02	Group 1	90.1±8.0	81.4±13.1	5.8±2.3	45.2±30.9
	Group 2	89.6±8.4	81.5±12.9	5.6±2.2	43.6±30.2
	Group 3	90.7±7.7	81.3±13.3	6.2±2.5	46.9±31.8
	Group 4	90.7±7.7	81.6±12.9	6.1±2.4	45.8±31.1
	Group 5	90.3±7.9	81.3±13.3	5.7±2.2	44.7±30.8
	Group 6	89.8±8.2	81.3±13.3	5.9±2.3	44.9±30.7
	Group 7	90.3±7.8	81.4±13.0	6.1±2.2	46.5±31.2
0.03	Group 1	90.0±8.1	81.6±12.9	5.7±2.3	44.8±30.9
	Group 2	89.6±8.4	81.9±12.8	5.5±2.2	43.2±30.2
	Group 3	90.6±7.7	81.3±13.1	6.1±2.4	46.1±31.4
	Group 4	90.5±7.7	81.6±13.0	6.1±2.4	45.2±30.7
	Group 5	90.3±7.9	81.4±13.2	5.6±2.2	43.9±30.4
	Group 6	89.7±8.2	81.5±13.0	5.9±2.4	44.5±30.6
	Group 7	90.1±7.8	81.6±12.7	6.1±2.3	46.2±31.5
0.04	Group 1	89.9±8.1	81.7±12.9	5.7±2.3	44.7±30.9
	Group 2	89.4±8.4	81.5±13.1	5.5±2.2	43.1±30.2
	Group 3	90.5±7.7	81.6±13.1	6.0±2.4	46.2±31.6
	Group 4	90.4±7.8	81.5±13.1	6.0±2.4	45.0±30.9
	Group 5	90.1±8.0	81.5±12.9	5.6±2.2	43.7±30.6
	Group 6	89.5±8.2	81.5±13.3	5.8±2.3	44.6±30.8
	Group 7	90.0±7.9	81.4±13.1	6.1±2.3	46.1±31.6

Table 6.14: Results of the t-tests comparing, for each group of discretizers, all configurations tested, using a confidence level of 0.05. The table shows for each configuration how many times it has been able to outperform significantly another method, and how many times it has been outperformed.

Group of disc.	Config.	Times outperforming	Times outperformed
Group 1	NoReinit	0	0
	Reinit 0.01	0	0
	Reinit 0.02	0	0
	Reinit 0.03	0	0
	Reinit 0.04	0	0
Group 2	NoReinit	0	6
	Reinit 0.01	1	1
	Reinit 0.02	1	0
	Reinit 0.03	3	0
	Reinit 0.04	2	0
Group 3	NoReinit	0	5
	Reinit 0.01	0	1
	Reinit 0.02	4	1
	Reinit 0.03	1	1
	Reinit 0.04	4	1
Group 4	NoReinit	0	4
	Reinit 0.01	0	1
	Reinit 0.02	2	0
	Reinit 0.03	2	0
	Reinit 0.04	1	0
Group 5	NoReinit	3	0
	Reinit 0.01	0	1
	Reinit 0.02	0	0
	Reinit 0.03	0	1
	Reinit 0.04	0	1
Group 6	NoReinit	0	7
	Reinit 0.01	1	0
	Reinit 0.02	2	0
	Reinit 0.03	2	0
	Reinit 0.04	2	0
Group 7	NoReinit	1	2
	Reinit 0.01	0	1
	Reinit 0.02	0	0
	Reinit 0.03	1	1
	Reinit 0.04	2	0

which shows better results without the reinitialize operator. Group 7 shows unclear results, although it seems that it can also benefit from reinitialize.

With the results of the t-tests and also using the accuracy averages for the case where there was no significant differences among the methods, we determine that the best setup for each group of discretization algorithms is the following:

Group 1 Configuration with reinitialize and probability 0.04

Group 2 Configuration with reinitialize and probability 0.03

Group 3 Configuration with reinitialize and probability 0.04

Group 4 Configuration with reinitialize and probability 0.03

Group 5 Configuration without reinitialize

Group 6 Configuration with reinitialize and probability 0.03

Group 7 Configuration with reinitialize and probability 0.04

Now it is time to determine which is the best group of discretization algorithms for *ADI*. Therefore, new t-tests will be performed comparing the best configuration of each group, described above. The results of these t-tests are in table 6.15. From these t-tests we can conclude that the best group is group 2, because it is the one outperforming other methods more times than any other group, and it is also the one being outperformed less times than any other group.

If we compare the performance of group 2 with the performance of using ADI with a single discretizer using the t-tests we see that group 2 is able to outperform another configuration (single discretizer in this case) 56 times, and it is being significantly outperformed only twice. If we look at the t-tests results of the single discretizers experimentation in table 6.9 we see that these results are better than the achieved by any single discretizer. This last comparison helps to confirm that the combination of discretizers is the most feasible way to use *ADI*, and that the set of discretizers in group 2 performs well and it is also very robust. Now it is time to compare the *ADI* representation with other alternatives.

Table 6.15: Results of the t-tests comparing the best setup of each group of discretizers, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Total
Group 1	-	0	0	1	1	1	1	4
Group 2	0	-	1	1	2	1	3	8
Group 3	0	0	-	0	2	0	1	3
Group 4	0	0	1	-	2	0	2	5
Group 5	0	1	1	1	-	1	1	5
Group 6	1	0	2	1	2	-	2	8
Group 7	0	0	0	0	1	1	-	2
Total	1	1	5	4	10	4	10	

6.6.3 Comparing ADI to two representations handling directly with real values

For this final comparison two knowledge representations have been selected:

Unordered bounds intervals representation (UBR) This representation uses rules with totally conjunctive predicates. The term assigned to each attribute in the rule is a real-valued interval codified with two real values, the bounds of the interval. The position of the lower and the upper bound is not defined, the lower of the two real values is the lower bound, the higher is the upper bound, like the ones used in (Stone & Bull, 2003). The representation is extended with two bits per attribute, which define if the lower or the upper bound are relevant. If one of these two bits declares its assigned bound as irrelevant, the test in the rule is converted to a "greater than" or "less than" type of test. If both bits are set to true, the test is always true, declaring the attribute irrelevant. In initialization these two bits have a 50% probability of being true, and the initial intervals have a size ranging from 25% to 75% of the attribute domain. Mutation flips the state of the bits or adds/subtracts a random offset from the bounds of the interval. The crossover operator is unchanged.

Instance set representation This representation (Llorà & Garrell, 2001a) evolves a set of synthetic prototypes, that act as the core of a 1-nearest neighbour classifier. Euclidean distance function is used. Mutation adds or subtracts a random offset from the attribute value, crossover is unchanged.

These are not the only real-valued representations existing in the literature (Cordón, Herrera, Hoffmann, & Magdalena, 2001; Llorá & Wilson, 2004), we selected only the two options described above because they are simple and easy to configure, and have a similar genetic representation (especially compared knowledge representations evolving decision trees). Also, in this comparison we do not include other non-evolutionary learning systems. The aim of these tests is to seek how suitable these knowledge representations are in the framework of our system. That is, how compatible the bias introduced by the representation and the one introduced by the learning system are. A comparison of *GAssist* with some other non-evolutionary learning systems appears in chapter 9.

Table 6.16 shows the results of the experimentation with the two selected real-valued knowledge representations. This table also includes two configurations of *ADI*, one using the best single discretization algorithm (*ID3*) and another one using group 2 of discretizers. For the sake of simplicity, we used common parameters for all configurations.

These results show how, on average, the best knowledge representation is the one evolving a set of instances. However, it has a computational cost which is four times higher than the cost of *ADI*. Also, this high accuracy does not translate into robustness. If we compute the t-tests over these results, shown in table 6.17, we can see that the instance set representation is the one outperforming most times, but it has been outperformed one more time than *ADI-Group2*. These results are a natural consequence from the difference in representation bias between the instance knowledge representation and all the other three (axis-parallel) representations. It is normal that this representation can achieve better results in some datasets, while it performs worse in some others. In general we can consider its performance, in the framework of *GAssist* as good, but given that it is not more robust than *ADI-Group2* and that it has a computational cost 4 times higher, we can consider that *ADI-Group2* is also a very good choice; it sacrifices some accuracy but it is much faster.

On the other hand, the performance of the representation codifying real-valued intervals can be considered as somewhat poor. Its only good quality is that it runs faster than *ADI*, but looking at the training accuracy we can see that this representation, as it is, has some problems exploring the real-valued search space. The huge reduction in the size of the search space introduced by the discretization process appears to be beneficial in this case.

Table 6.16: Results of the tests comparing ADI to two real-valued representations.

Dataset	Technique	Training acc.	Test acc.	#rules/#prototypes	Run-time (s)
bal	ADI-ID3	85.8±0.7	78.7±4.4	10.7±2.2	36.8±5.2
	ADI-Group2	85.1±0.7	79.3±3.5	9.7±1.9	34.2±5.5
	UBR	88.2±1.0	80.1±4.3	13.5±2.5	26.5±4.8
	Instances	92.2±0.2	90.0±2.0	35.5±15.6	129.7±22.0
bpa	ADI-ID3	84.9±1.6	65.3±7.4	9.6±1.8	42.3±6.7
	ADI-Group2	79.3±1.5	66.8±6.9	7.5±1.3	38.3±5.6
	UBR	77.6±3.4	62.8±7.1	8.9±1.3	25.8±4.0
	Instances	83.6±1.7	66.2±7.7	27.7±8.1	94.1±13.9
gls	ADI-ID3	82.4±1.9	69.6±9.1	7.3±1.1	85.0±5.2
	ADI-Group2	79.3±2.3	67.3±9.8	6.7±0.8	84.3±6.0
	UBR	78.3±2.2	67.4±9.4	8.5±1.1	55.5±4.6
	Instances	88.1±2.3	67.5±9.9	30.5±5.1	330.4±55.2
h-s	ADI-ID3	91.5±1.0	79.8±7.2	7.3±1.2	24.7±2.0
	ADI-Group2	90.7±1.0	81.1±7.7	7.0±0.9	24.4±2.5
	UBR	91.4±1.4	79.2±7.6	8.7±1.3	18.3±2.4
	Instances	92.2±1.1	81.2±6.8	13.4±3.9	54.1±8.5
ion	ADI-ID3	97.7±0.9	90.1±5.5	3.7±1.1	62.2±11.5
	ADI-Group2	97.2±0.5	92.3±5.0	2.1±0.3	54.5±8.3
	UBR	96.3±2.2	90.4±4.9	4.8±1.1	37.3±6.5
	Instances	98.7±0.5	90.8±5.0	18.0±5.6	390.3±64.8
irs	ADI-ID3	98.2±0.8	95.0±5.8	3.6±0.7	3.8±0.5
	ADI-Group2	97.9±0.9	94.4±5.9	3.5±0.6	4.1±0.5
	UBR	97.9±0.9	94.4±5.6	3.8±0.7	2.7±0.6
	Instances	99.5±0.4	95.2±6.0	4.2±0.9	6.0±0.8
lrn	ADI-ID3	76.9±1.0	69.2±5.2	9.2±1.9	88.1±8.2
	ADI-Group2	76.0±0.8	69.9±4.7	7.6±1.2	90.8±8.2
	UBR	75.9±1.2	69.0±5.0	8.6±1.5	85.2±13.7
	Instances	78.7±1.1	66.5±4.9	67.2±17.0	510.4±112.4
mmg	ADI-ID3	87.6±1.5	64.9±10.8	7.1±1.1	50.6±6.6
	ADI-Group2	83.4±1.3	68.7±11.2	6.5±0.7	46.0±4.4
	UBR	81.4±2.5	64.8±10.3	6.9±0.9	31.7±4.1
	Instances	79.9±1.8	63.4±10.8	8.4±1.7	100.8±9.5
pim	ADI-ID3	84.2±0.8	73.7±4.8	9.5±1.9	104.6±15.5
	ADI-Group2	82.9±0.8	74.9±4.9	7.6±1.4	103.1±11.6
	UBR	82.4±1.3	74.0±4.6	10.3±1.2	64.5±12.2
	Instances	84.6±0.9	74.8±5.3	37.9±11.7	379.4±62.4
thy	ADI-ID3	99.1±0.7	92.9±5.3	5.4±0.6	8.5±0.9
	ADI-Group2	99.0±0.5	92.1±5.4	5.3±0.5	8.5±0.9
	UBR	97.3±1.1	91.5±5.8	5.4±0.6	5.8±0.8
	Instances	99.6±0.5	95.7±3.8	5.3±0.6	13.7±1.5
wbcd	ADI-ID3	98.5±0.3	95.7±2.4	3.5±0.7	15.7±2.1
	ADI-Group2	97.9±0.4	96.0±2.3	3.0±0.7	14.7±1.8
	UBR	98.1±0.4	95.4±2.5	4.4±1.4	12.9±3.0
	Instances	98.4±0.3	96.0±2.4	6.7±4.1	76.3±16.7
wdbc	ADI-ID3	98.9±0.4	94.2±3.0	5.2±1.2	69.9±15.3
	ADI-Group2	98.0±0.5	94.0±3.1	3.9±0.7	54.9±9.3
	UBR	97.4±0.8	93.9±3.0	5.0±1.2	36.9±6.6
	Instances	99.1±0.3	96.6±2.5	5.4±2.4	189.5±53.9
wine	ADI-ID3	99.7±0.4	93.7±5.5	3.9±0.8	15.0±1.7
	ADI-Group2	99.9±0.3	93.8±5.6	3.2±0.5	14.3±1.3
	UBR	99.4±0.5	92.3±6.1	4.6±1.1	10.2±1.0
	Instances	100.0±0.1	96.3±4.6	3.7±0.7	30.7±4.7
wpbc	ADI-ID3	92.5±1.7	73.2±8.8	6.7±1.5	43.7±9.3
	ADI-Group2	87.2±2.1	76.3±8.1	3.3±1.1	32.4±4.2
	UBR	88.7±1.8	72.9±8.5	7.7±1.4	28.4±2.7
	Instances	89.4±1.8	75.8±7.9	7.6±2.8	91.5±9.9
Ave.	ADI-ID3	91.3±7.4	81.1±11.5	6.6±2.4	46.5±30.7
	ADI-Group2	89.6±8.4	81.9±12.8	5.5±2.2	43.2±30.2
	UBR	89.3±8.6	80.6±13.4	7.2±2.6	31.6±22.6
	Instances	91.7±7.6	82.6±14.1	19.4±17.7	171.2±157.1

Table 6.17: Results of the t-tests comparing the ADI representation with two alternative knowledge representations handling directly real values, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	ADI-ID3	ADI-Group2	UBR	Instances	Total
ADI-ID3	-	1	1	2	4
ADI-Group2	4	-	6	3	13
UBR	1	0	-	1	2
Instances	6	4	8	-	18
Total	11	5	15	6	

6.7 Discussion and further work

In order to decide which were the best single discretizers and, after, the best groups of discretizers we have used the same set of 14 datasets. In order to validate if the group of discretizers that was determined by all the experimentation to be the best is really competent, we have to test it over a much larger set of problems.

The rest of this section deals with the dynamics of the *ADI* representation. We have seen, first from the results, and then looking at the proportions of each discretizer in the population that *ADI* was not able to discover the most suitable discretizers in all datasets used in the experimentation. In order to fix this problem, we introduced the reinitialization operator, which performs a very drastic and destructive action. Playing with the probability controlling the operator we have been able to use it with almost no destructive effects. However, there may be other ways to apply the operator, by either a different probability setting policy, biasing the random selection of the new discretizer or perhaps by selecting deterministically the attributes that are going to be reinitialized based on some criteria that studies the performance of the attribute.

Nevertheless, instead of fixing the problem of the discretizers disappearing from the population, we should avoid it. Looking at the previous chapter, dealing with default rules, we have developed a niching process to allow the system to choose correctly a default class. In some sense, here we are in the same situation. We should preserve all discretizers until the system is able to decide by itself. The problem is that performing a niching process over attributes is difficult, because an individual can contain in most datasets hundreds of attributes. If we use traditional recombination operators, this task is very difficult. Maybe, in order to deal with this question we should transform our system into a kind of estimation of distribution algorithm

(Larranaga & Lozano, 2002), that samples individuals from a model. If we can construct individuals piece by piece, it is much easier to implement the needed niching mechanisms.

6.8 Summary

In this chapter we have described our contribution in the area of representations for real-valued attributes in genetic-base machine learning. This contribution is a representation, called *adaptive discretization intervals* (ADI) rule representation, that evolves rules that can use multiple discretization algorithms, letting the evolution choose the correct discretization for each rule and attribute. Also, the intervals defined in each discretization can split or merge among them through the evolution process, reducing the search space where it is possible.

The chapter started by describing the basic mechanisms of the representation, then the illustration of the behavior of this basic system led to identifying some problems that motivated the development of a new operator, also described in depth.

Next came a large number of experiments. First testing the representation using only one discretizer at the same time. Several proposals of groups of discretizers were made, using the performance of these single-discretizer tests to determine these groups. The groups of discretizers were tested in several setups of the representation. These tests were useful to learn which was the most suitable configuration for each group of discretizers (and, as an extension, for the families of discretizers represented by the groups). The tests validated that the combination of discretizers is a better approach than using only one discretization algorithm, because we obtain a system with slightly higher accuracy but which is much more robust.

The ADI representation with the best single and composed set of discretizers was compared against two alternative knowledge representations handling directly real values, one evolving real-valued intervals and one evolving a set of instances as a core of a nearest neighbour classifier. The comparison with the UBR representation showed that *ADI* performs better and is more robust. The comparison against the representation generating prototypes showed that, although the accuracy of *ADI* is smaller, it manages to be equally robust, and it is much faster. This leads to the conclusion that both representations, *ADI* as well as the instance-set-generation are equally good for using inside the framework of *GAssist*

Finally we described some further work, focusing especially on how to fix or solve the problem of being able to guarantee that we can choose correctly the discretizer that is going to be used in the representation.

Chapter 7

Windowing techniques for generalization and run-time reduction

Windowing methods are useful techniques to reduce the computational cost of Pittsburgh-style genetic-based machine learning techniques. Also, if used properly, they can also be used to improve the classification accuracy of the system. This chapter describes the research done on these techniques. After describing the development of the windowing method that we use, called ILAS *(incremental learning by alternating strata), we start by studying its behavior on small datasets, proposing a model of the maximum run-time reduction degree achievable without performance decrease. Also, a run-time model is developed. The models lead us to propose several strategies for the use of* ILAS*. On small datasets they are used to check how can we maximize the performance of the system with minimum overhead, or maintain this performance with significant run-time reduction. On large datasets the objective is to achieve good performance while reducing very significantly (sometimes one or more order of magnitude) the run-time.*

The chapter is structured as follows: First, section 7.1 will contain an introduction to the windowing techniques studied, followed by section 7.2 containing some related work. Next, section 7.3 will show an historical description of the development process of ILAS *illustrated with some past results that will show the motivation for the rest of the chapter. Section 7.4 will describe the models developed to study the behavior of* ILAS*, followed by the experimental study of the performance of* ILAS *in small datasets in section 7.5. Next, section 7.6 will contain the corresponding experimentation on large datasets, section 7.7 will provide a discussion about the studied windowing technique and some further work. Finally, section 7.8 will summarize the chapter.*

CHAPTER 7. WINDOWING TECHNIQUES FOR GENERALIZATION AND RUN-TIME REDUCTION

7.1 Introduction

One of the traditional drawbacks associated to the GBML systems using the Pittsburgh approach is the high computational cost associated. The reason of this cost, as usual in GAs, is the fitness computation. In this case because computing the fitness of each individual means using it to classify the whole training set.

We can primarily reduce the cost of these fitness computations by either: (a) decreasing complexity of the individuals or (b) decreasing the dimensionality of the domain to be classified (there are other methods such as fitness inheritance or informed competent operators but they may affect the whole *GA* cycle). The former methods are usually referred to as *parsimony* pressure methods (Soule & Foster, 1998). The latter methods are either characterized as feature selection methods, reducing the number of problem attributes, or as incremental learning or windowing methods, reducing the number of training instances per fitness computation.

In previous work (Bacardit & Garrell, 2003d; Bacardit & Garrell, 2003b), we empirically tested some training set reduction schemes. These schemes select a training subset to be used for fitness computation, changing the subsets through the iterations of the *GA* process. Thus, being a kind of windowing process. Our previous results showed that the techniques achieved the run-time reduction objective with no significant accuracy loss. Sometimes, test accuracy actually increased, indicating knowledge generalization pressures that may alleviate over-fitting. The technique that obtained the best performance is called *ILAS* (incremental learning by alternating strata). This technique divides the training set into n strata, which are the subsets used in the fitness computations. It changes the used strata at each iteration, using a round-robin policy. The new research being described in this chapter will focus exclusively on *ILAS*.

We use the "Incremental Learning" term to name this scheme in the sense that the individuals of the population (rule sets) are refined through the iterations based on some knowledge (instances) that is presented to these individuals in small increments (strata). It should not be confused with other kinds of learning methods in which the model (e.g. a rule set) is constructed by merging partial modules learned with subsets of the training set (Giraud-Carrier, 2000).

In our previous work several open questions remained. From a run-time reduction perspective, we were interested in deriving a model of the maximal learning time reduction while avoiding significant accuracy loss. From a learning perspective, we were interested in the learning time reduction that maximizes accuracy in the system, given a constant run time. In order to achieve the latter objective, the development of a run-time cost model was needed. In (Bacardit, Goldberg, Butz, Llorá, & Garrell, 2004) the development of these models started,

based on some synthetic datasets.

These models were a framework for future experimentation on *ILAS*, which is described in this chapter. The experimentation is split in two parts: *ILAS* focused on small datasets and *ILAS* focused on large datasets. On small datasets, maximizing the performance is the main objective. On large problems the objective is to achieve the maximum run-time reduction possible without significant accuracy loss.

7.2 Related work

Following the classification of training set reduction techniques described in section 2.6, the *ILAS* windowing scheme is a modified learning algorithm. In previous work (Bacardit & Garrell, 2003b), reproduced in next section, the prototype selection approach was tested, using a Case-Base Maintenance technique called *ACCM* (Salamó & Golobardes, 2003) which is based on Rough Sets theory. The results obtained, however, were poor, both in accuracy and computational cost. The prototype selection way has a very high risk of introducing an irreversible negative bias into the system. This bias is smaller in *ILAS* because all instances of the training set, through the learning process, are used. Moreover, one of the two models of *ILAS* described in this chapter has as an objective to analyze this bias.

The *GABIL* system (DeJong, Spears, & Gordon, 1993), which is the original inspiration of *GAssist*, is an example of wrapper method. As said in section 3.5 this approach does not seem to be very suitable for real-world problems. Also, the wrapper approach is not very suitable for a *GBML* system (especially a Pittsburgh one) because the base computational cost of running a full *GA* learning process is already high. Even if we can reduce considerably the training set, the overhead of the *GA* cannot be ignored.

From a GBML perspective, described in section 3.5, the windowing techniques studied in this chapter can be considered as generation-wise, and performing a sampling process controlled by age exclusively.

7.3 The development process of the ILAS windowing technique and previous results

This section describes the development process that led to the proposal of the windowing technique, *ILAS*, studied in this chapter. This process consisted of the sequential proposal of three windowing techniques, each of them fixing the problems of the previous one. The third technique is *ILAS*, the windowing method used in the rest of the chapter. All techniques varied the subset of training examples used for the fitness computations through the *GA* iterations. The criteria to select this subset (*stratum*) and also when and how often this stratum was changed defined each of them. The section also reproduces the initial results achieved with these techniques.

7.3.1 Basic Incremental Learning (BIL)

Our first proposal simply divides the training set and the *GA* iterations in *N* uniform parts, and uses the training stratum 1 for the first stage of iterations, training stratum 2 for the second stage of iterations, and so on. The last iteration of the learning process will use the whole training set because we need to select the individual which will provide the final theory, and it should be a good solution for all the training data, not only for the final stratum. The original examples are reordered to maintain, for each stratum, the same class distribution that exists in the whole training set. This scheme is represented by the code in figure 7.1. The procedure to reorder the examples in strata with equal class distribution is described in figure 7.2. The aim of this procedure is to generate strata reducing the introduced bias of the stratification as much as possible.

A quick test was done to check the performance of this *BIL* scheme. The test used the *bps* problem which is detailed in section 4.3, the configuration of GAssist for all the tests in this section is summarized in table 7.1. Tests with 2, 3 and 4 strata were done, and the results are shown in table 7.2. The meaning of the column labeled *speedup* is the ratio between the non-incremental and incremental schemes run times. This speedup cannot be compared to the speedup measure of algorithm parallelization because we are modifying the algorithm. For this reason, we can find speedups higher than the number of strata used.

The speedups obtained using 2 and 3 strata were as expected, and a surprise for the 4 strata test. However, there was an accuracy (percentage of correctly classified examples) loss for all the incremental tests. These results bring up a question: Is this decrease normal? Looking at the systems described in the related work section we can see reports of both performance increase and decrease. Thus, we have to explore why we have a performance decrease and

Figure 7.1: Code of the Basic Incremental Learning scheme

Procedure Basic Incremental Learning
Input : $Examples, NumStrata, NumIterations$
Initialize GA
$Examples =$ ReorderExamples($Examples$,$NumStrata$)
$Iteration = 0$
$StrataSize = size(Examples)/NumStrata$
While $Iteration < NumIterations$
  If$Iteration = NumIterations - 1$
    $TrainingSet = Examples$
  Else
    $CurrentStratum = int(Iteration \cdot NumStrata/NumIterations)$
    $TrainingSet=$ examples from $Examples[CurrentStratum \cdot StratumSize]$
      to $Examples[(CurrentStratum + 1) \cdot StratumSize]$
  EndIf
  Run one iteration of the GA with $TrainingSet$
  $Iteration = Iteration + 1$
EndWhile
Output : Best set of rules from GA population

Figure 7.2: Code of the strata generation with equal class distribution

Procedure ReorderExamples
Input : $Examples, NumStrata$
$NumClasses =$ number of classes existing in $Examples$
$ListsOfExamples = Vector[NumClasses]$ of sets of examples
$Strata = Vector[NumStrata]$ of sets of examples
ForEach $example$ in $Examples$
  $cls =$ class of $examples$
  Add $example$ to $ListsOfExamples[cls]$
EndForEach
$stratum = 0$
ForEach cls in $NumClasses$
  While size of $ListsOfExamples[cls] > 0$
    $example =$ remove a random example from $ListsOfExamples[cls]$
    Add $example$ to $Strata[stratum]$
    $stratum = (stratum + 1)\ mod\ NumStrata$
  EndWhile
EndForEach
$NewExamples =$ Merge $Strata$ into a single set putting in order the examples of each strata
Output : $NewExamples$

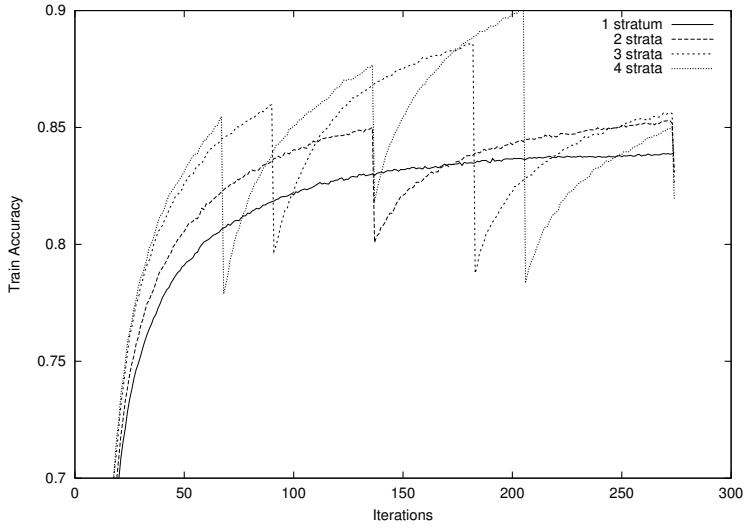
Table 7.1: Settings of GAssist for windowing experiments reported in section 7.3

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	300
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1000
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	disabled
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
Hierarchical selection operator	
Iteration of activation	25
Threshold	0.01

Table 7.2: Quick test of the *BIL* scheme

Strata	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	79.5%	-0.7%	1.92
3	79.0%	-1.2%	2.96
4	78.9%	-1.3%	4.81

Figure 7.3: Accuracy evolution for the Basic Incremental Learning scheme for the *bps* problem



whether it can be fixed. The next scheme will try to solve this problem.

7.3.2 Basic Incremental Learning with a Total Stratum (BILTS)

The reason of the performance decrease of the *BIL* scheme can be seen by looking at the accuracy evolution through the iterations. This evolution is represented in figure 7.3. At each stratum change, there is an accuracy loss because the knowledge of the current population is not enough to classify some of the new training examples.

Thus, new knowledge has to be learned. The process of adding new knowledge will modify some of the current rules, but it will also add new ones. This can lead to a situation where some of the good rules for the previous strata are not used anymore, and these rules are deleted by our *rule deletion* operator. The old useless knowledge is forgotten. Looking again at the graph in figure 7.3 we can see a performance hit at the end of the learning. This is due to

Figure 7.4: Code of the Basic Incremental Learning with Total Stratum scheme

Procedure Basic Incremental Learning with Total Stratum
Input : $Examples, NumStrata, NumIterations$
Initialize GA
$Examples =$ ReorderExamples($Examples$,$NumStrata$)
$Iteration = 0$
$StratumSize = size(Examples)/NumStrata$
While $Iteration < NumIterations$
    $CurrentStratum = int(Iteration \cdot (NumStrata + 1)/NumIterations)$
    If $CurrentStratum = NumStrata$
      $TrainingSeg = Examples$
    Else
      $TrainingSeg =$ examples from $Examples[CurrentStratum \cdot StratumSize]$ to $Examples[(CurrentStratum + 1) \cdot StratumSize]$
    EndIf
    Run one iteration of the GA with $TrainingSeg$
    $Iteration = Iteration + 1$
EndWhile
Output : Best set of rules from GA population

using the whole training stratum in the last iteration. All the forgotten knowledge could be useful again, but it is missing.

How can we fix the problem of forgetting good knowledge and, as a consequence, prevent the performance loss? Switching off the rule deletion operator is not a feasible solution because the population would grow without control, as was seen in chapter 8. A simple alternative is adding a stage at the end of the learning process where all the training examples are used. This scheme, called *Basic Incremental Learning with a Total Stratum (BILTS)* is represented by the code in figure 7.4.

We repeated the same quick tests done for the previous scheme, and its results can be seen in table 7.3. Figure 7.5 shows the graph of the accuracy evolution. The performance loss is smaller now and even there is performance increase using 4 strata. The speedup, obviously, is less than the obtained with the previous scheme, but it is still worthy.

With this second scheme we have fixed the loss of knowledge derived from a stratum change, but it has cost a decrease in the speedup of the system. Instead of fixing the loss of knowledge maybe we should prevent it and, therefore, avoid the stage of the learning using the whole training set.

Table 7.3: Performance of the *Basic Incremental Learning with Total Stratum* scheme

Strata	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	80.1%	-0.1%	1.50
3	80.0%	-0.2%	2.06
4	80.5%	+0.3%	2.93

Figure 7.5: Accuracy evolution for the Basic Incremental Learning with Total Stratum scheme for the *bps* problem

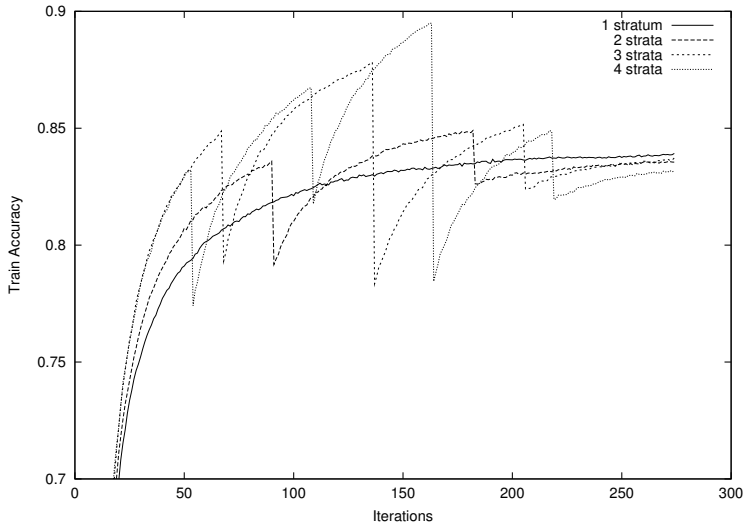


Figure 7.6: Code of the Incremental Learning with Alternating Strata scheme

Procedure Incremental Learning with Alternating Strata
Input : $Examples, NumStrata, NumIterations$
Initialize GA
$Examples =$ ReorderExamples($Examples$,$NumStrata$)
$Iteration = 0$
$StratumSize = size(Examples)/NumStrata$
While $Iteration < NumIterations$
  If $Iteration = NumIterations - 1$
    $TrainingSeg = Examples$
  Else
    $CurrentStratum = Iteration \ mod \ NumStrata$
    $TrainingSeg =$ examples from $Examples[CurrentStratum \cdot StratumSize]$ to $Examples[(CurrentStratum + 1) \cdot StratumSize]$
  EndIf
  Run one iteration of the GA with $TrainingSeg$
  $Iteration = Iteration + 1$
EndWhile
Output : Best set of rules from GA population

7.3.3 Incremental Learning with Alternating Strata (ILAS)

In order to prevent a knowledge loss we should use all the training examples with enough frequency to assure that all the knowledge of the individuals becomes useful and, as a consequence, it is not forgotten. Bringing this idea to the extreme, we propose another scheme called *Incremental Learning with Alternating Strata*, which changes the used stratum at each iteration. Its code can be seen in figure 7.6.

Again, we repeated the same quick test. Its results can be seen in table 7.4 and in figure 7.7. Some oscillations have appeared in the evolution of the incremental schemes. They are due to the fact that there are some strata which are easier than some others. By easier strata we understand strata that have, by chance, less noise, more uniform distribution of examples, etc. Now the results are really surprising for both the accuracy increase and the speedup. How can we explain the performance and speedup increase?

Our hypothesis is based on the behavior of the classifier system given our fitness function (squared training accuracy). This fitness function does not make differences whether training examples are learned (creating generalized rules) or memorized (creating specific rules). Using a non-incremental *GA*, the easiest way to increase the fitness is to memorize examples instead of learning them. Generality pressure methods can reduce this problem, but not completely fix it.

Figure 7.7: Accuracy evolution for the Incremental Learning with Alternating Strata scheme for the *bps* problem. Sampling of the accuracy every 5 iterations

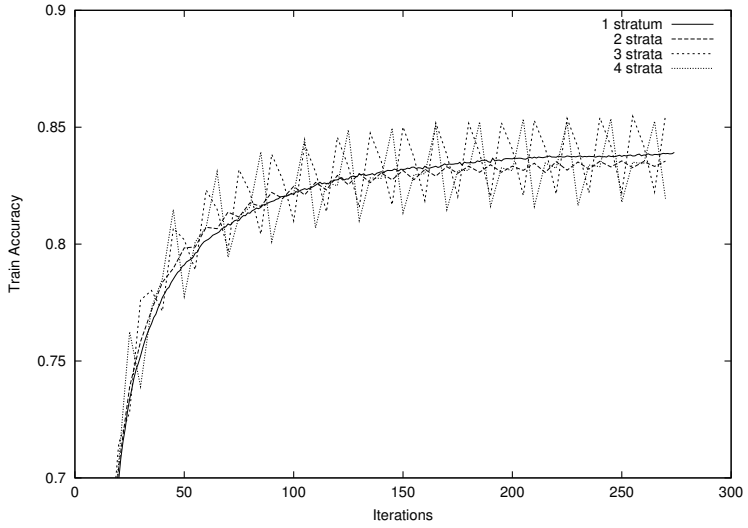


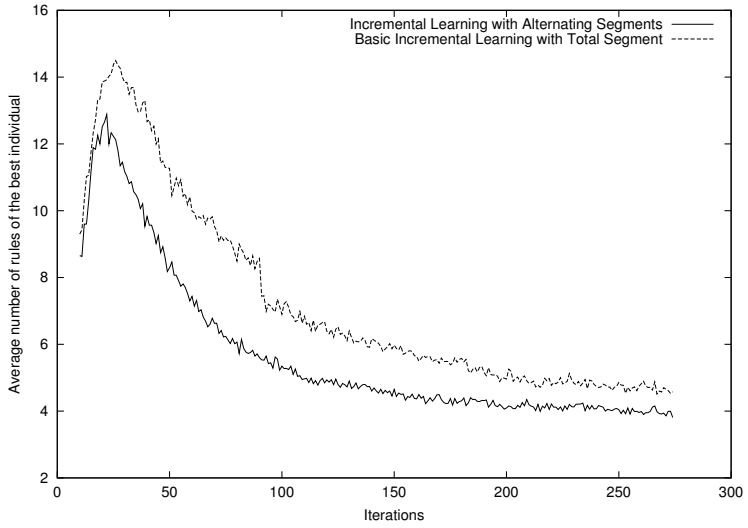
Table 7.4: Performance of the *Incremental Learning with Alternating Stratum* scheme

Strata	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	80.5%	+0.3%	2.05
3	80.8%	+0.5%	3.6
4	80.4%	+0.2%	6.01

The *ILAS* scheme changes dramatically the environment of the GA population: The adaptation of the individuals to the high frequency changes of the training set consists of learning the examples instead of memorizing them, because the chances of surviving in the population become higher. The use of the rule deletion operator implicates that all the rules have to, at least, match one example in every stratum. This fact penalizes specific rules.

Therefore, more generalized solutions are generated and these solutions have more chances of having a good test accuracy. Also, more generalized solutions usually mean smaller solutions. This is the reason of the speedup increase and can be seen in figure 7.8.

Figure 7.8: Evolution of the number of rules for the incremental learning schemes with two strata in the *bps* problem



7.3.4 Some previous results on ILAS

Table 7.5 show previous results (Bacardit & Garrell, 2003b) of the *ILAS* scheme applied to some datasets. The first three datasets are small (less than 1000 instances), while the rest of datasets are of medium size (ranging from 6435 to 10992 instances). For the small datasets we tested *ILAS* using 2, 3 and 4 strata and for the medium datasets we used 5, 10 and 20 strata. The *ILAS* scheme is compared to the standard non-windowed system, labeled *NON*. The table includes results for accuracy and speedup.

The datasets shown in table 7.5 exhibit different behavior patterns. The runs in the small datasets show that accuracy increases in *wbcd* and *ion* when using *ILAS* but not in *pim*. Moreover, the maximum accuracy for *wbcd* is achieved using 3 strata, while in *ion* it is achieved using 4 strata. In the large datasets, a larger number of strata slightly decreases accuracy while strongly improving computational cost. Thus, using *ILAS* can be beneficial in two aspects: either an actual accuracy increase may be achieved in small datasets while achieving significant run-time reduction or stronger run-time reduction can be achieved while only slightly decreasing accuracy. We are interested in how *ILAS* may be applied to achieve optimal results focusing on learning time and learning accuracy with respect to the number of strata s.

In the next section, we first develop a model of what makes a dataset hard for *ILAS*. Once we achieve this objective and we know which is the maximum number of strata we can use for

Table 7.5: Previous results of ILAS and plot of individual size reduction. Dat=dataset, Sch = windowing scheme, Acc=Test accuracy, Spe=Speedup

Dat	Sch	Acc	Spe	Dat	Sch	Acc	Spe
wbcd	NON	95.6%	—	pen	NON	79.9%	—
	ILAS2	95.9%	2.72		ILAS5	79.9%	5.18
	ILAS3	96.0%	4.63		ILAS10	79.4%	10.37
	ILAS4	95.8%	5.70		ILAS20	78.9%	20.44
ion	NON	89.5%	—	sat	NON	79.9%	—
	ILAS2	90.2%	2.72		ILAS5	79.9%	4.73
	ILAS3	90.6%	4.63		ILAS10	79.4%	9.04
	ILAS4	91.0%	5.70		ILAS20	78.9%	16.54
pim	NON	75.2%	—	thy	NON	93.6%	—
	ILAS2	74.8%	2.67		ILAS5	93.7%	5.20
	ILAS3	74.6%	4.41		ILAS10	93.6%	9.84
	ILAS4	74.0%	5.85		ILAS20	93.5%	18.52

a dataset, we can decide with how many strata *ILAS* should be applied to a given problem.

If the dataset is small, we can use *ILAS* to improve the accuracy performance of the system regardless of the run-time reduction. Therefore, we need to predict how many GA iterations of *ILAS* use make it has the same run-time as the non-windowed system. On the other hand, if we are dealing with a medium or large dataset where the run-time reduction is the main concern, we have to predict how many iterations we can perform given a maximum run time. For both kinds of datasets we need to develop a run-time model.

7.4 The behavior models of ILAS

This section presents our models for the hardness of a dataset for *ILAS* and a computational cost model. The models are crucial for estimating the optimal ILAS settings for a given problem.

7.4.1 What makes a problem hard to solve for ILAS?

We start our study focusing on the multiplexer (Wilson, 1995) family of problems—a widely used kind of dataset with a well-known model. Our first step is to perform experiments determining how many iterations are needed to achieve 100% accuracy (convergence time)

Table 7.6: Settings of GAssist for windowing experiments reported in section 7.4

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	300
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	disabled
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Probability of Reinitialize (begin,end)	(0.02,0)
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

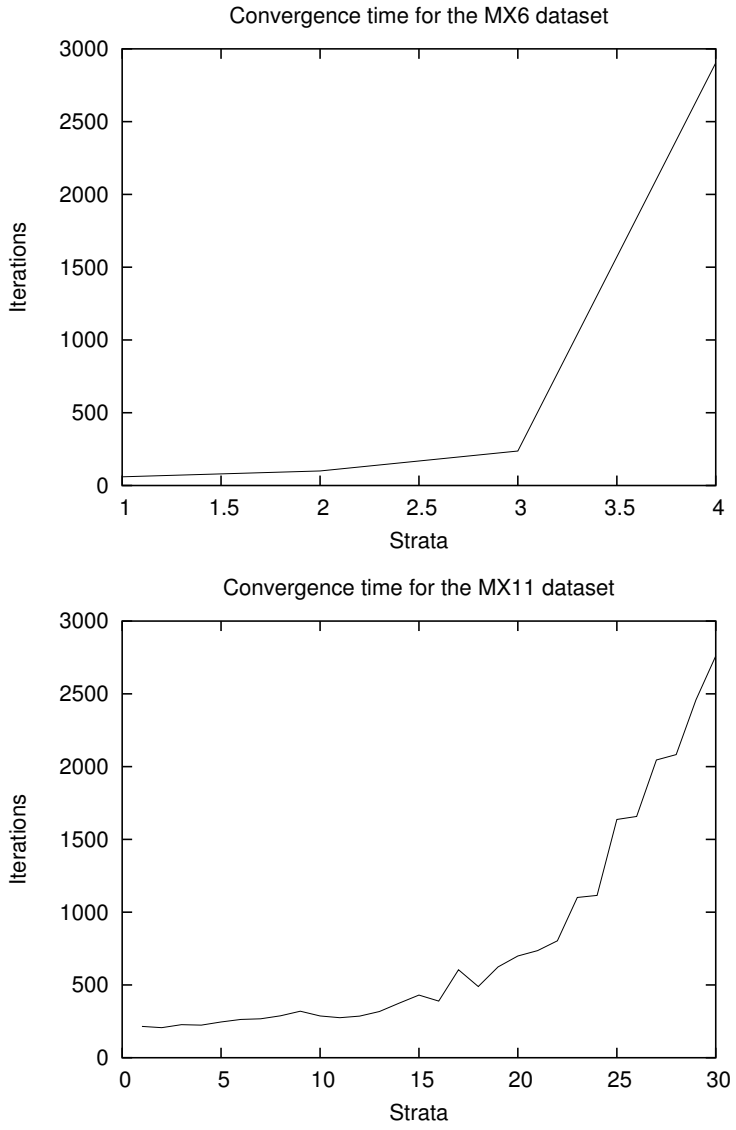
using the *ILAS* scheme for a given number of strata. The results of the experiments for the 6 (MX6) and 11 (MX11) bits multiplexer are shown in Figure 7.9. The plots are averaged over 50 independent runs. The settings of these tests are summarized in table 7.6

For both datasets we can see that the convergence time increases with the number of strata in an exponential way. Before a certain break point, the first part of the curve can be approximated by a linear increase. This break point is the maximum number of strata that is worth using in a dataset.

Intuitively we may suspect that after the break point the strata tend to miss-represent the whole training set causing learning disruptions. Since we know the optimal rule size in the multiplexer dataset, we are able to estimate how representative a strata may be. In the case of MX6 we have 8 rules, each rule covering 8 instances. In the case of MX11 we have 16 rules, each one covering 128 instances[1]. Only by observing these numbers it is quite easy to see

[1]Figure 5.1 in chapter 5 shows the optimal rule set for the MX-11 domain

Figure 7.9: Convergence time for the MX6 and MX11 datasets



that MX6 has a higher risk of having one of these rules unrepresented in some strata, which translates into having a break point at strata 3 (as seen in figure 7.9).

In order to predict the break point, we calculate the probability of having a particular rule (which corresponds to a sub-solution) unrepresented in a certain strata. We can approximate

this probability supposing uniform sampling with replacement:

$$P(\text{unrepresented rule}/s) = (1-p)^{\frac{\mathcal{D}}{s}} \tag{7.1}$$

where p denotes the probability that a random problem instance represents a particular rule, $\mathcal{D}$ is the number of instances in the dataset and s is the number of strata. The probability essentially estimates the probability that a particular rule is not represented by any problem instance in a strata.

A general probability of success (requiring that no rule is unrepresented) of the whole stratification process can now be derived using the approximation $(1-\frac{r}{s})^s \approx e^{-r}$ twice to simplify:

$$P(success/s) = (1 - P(\text{unrepresented rule}/s))^{rs} \tag{7.2}$$

$$P(success/s) = e^{-rs \cdot e^{-\frac{pD}{s}}} \tag{7.3}$$

where r denotes the number of rules. The derivation assumes that p is equal for all rules which is the case for our experimental verification below. If p differs, a derivation of success is still possible but the closed form is not derivable anymore.

The model is experimentally verified for *MX6* and *MX11* in figure 7.10. The experimental plot is the average of performing 2500 stratification processes and monitoring when there was an unrepresented rule. We can observe that the theoretical model is quite close to the experimental data, although it is slightly more conservative.

If we overlap this probabilistic model with the convergence time curve we can see that the exponential area of the convergence time curve starts approximately when the success probability drops below 0.95. We show this observation in figure 7.11 for the MX6 and MX11 and also for two versions of MX6 that have 2 (MX6_2) and 4 (MX6_4) additional redundant bits, thus being more robust to the stratification process than MX6. We can predict approximately the break point, achieving one of the objectives of this section.

Figure 7.10: Probability of stratification success. Verification of model with empirical data

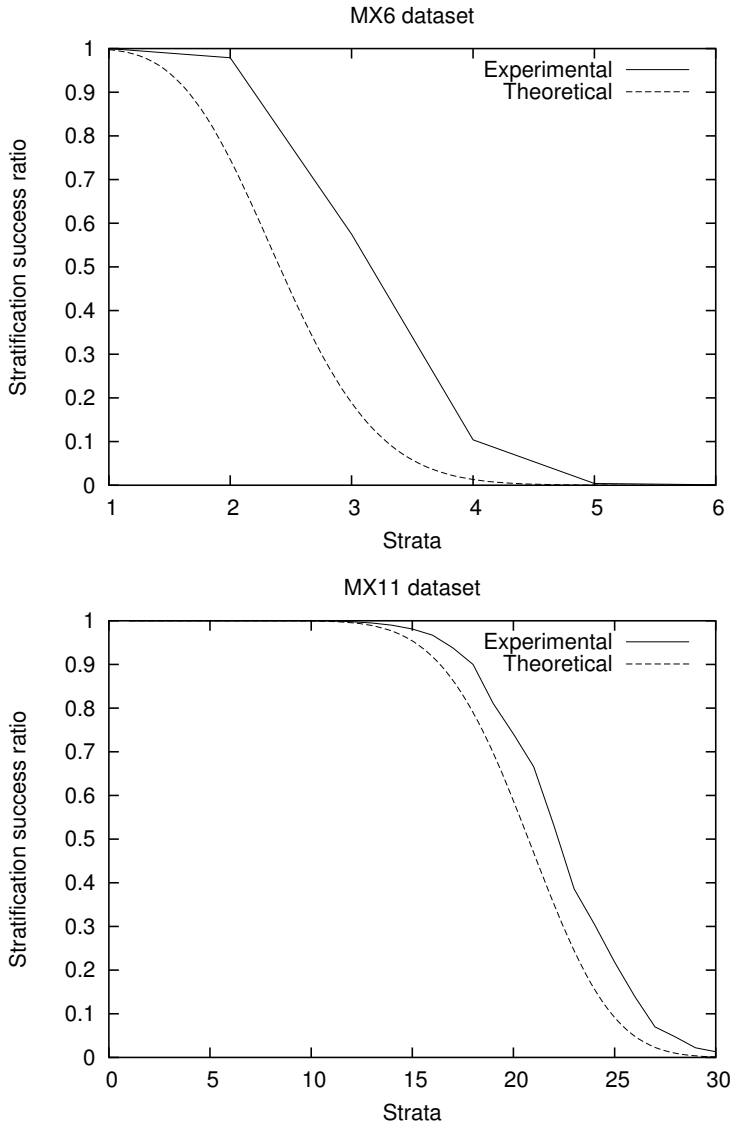
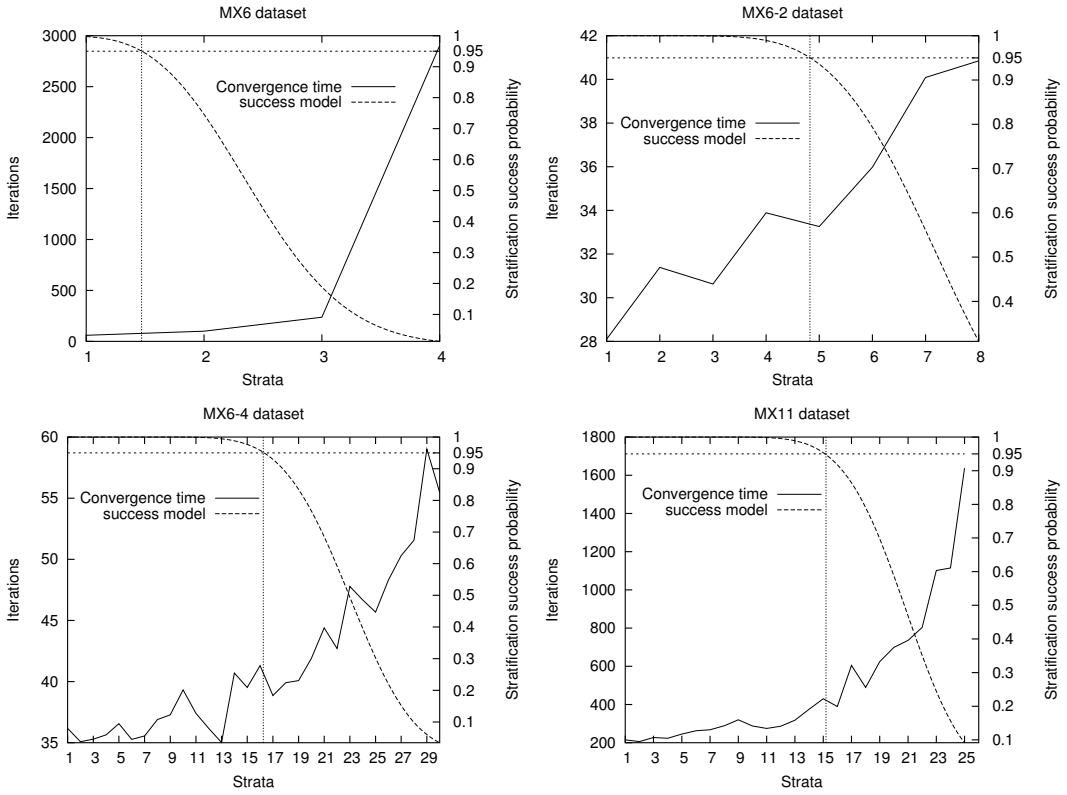


Figure 7.11: Comparison of the convergence time and the probability of stratification success. The vertical scale for left hand side of plots corresponds to iterations of convergence time. The scale for right hand side is the probability of stratification success (equation 7.3). The vertical and horizontal lines mark the 0.95 success point



7.4.2 Cost model of ILAS

The second objective of this section is the development of a run-time model. Assuming constant run time per iteration, we can model the run-time of the system by

$$T = \alpha \cdot it \tag{7.4}$$

where T denotes the total time of the learning process, α the time per iteration and it the number of iterations. This supposition may seem to contradict the rationale for the speedup results reported in section 7.3. The experimental setup used here has changed in one crucial parameter: the minimum number of rules of the rule deletion operator, which has increased considerably. While the number of alive rules in the individuals decreases when we increase the used strata, maintaining the implicit generalization pressure reported previously, the total number of rules per individuals is maintained approximately constant, which allows us to assume the supposition stated above.

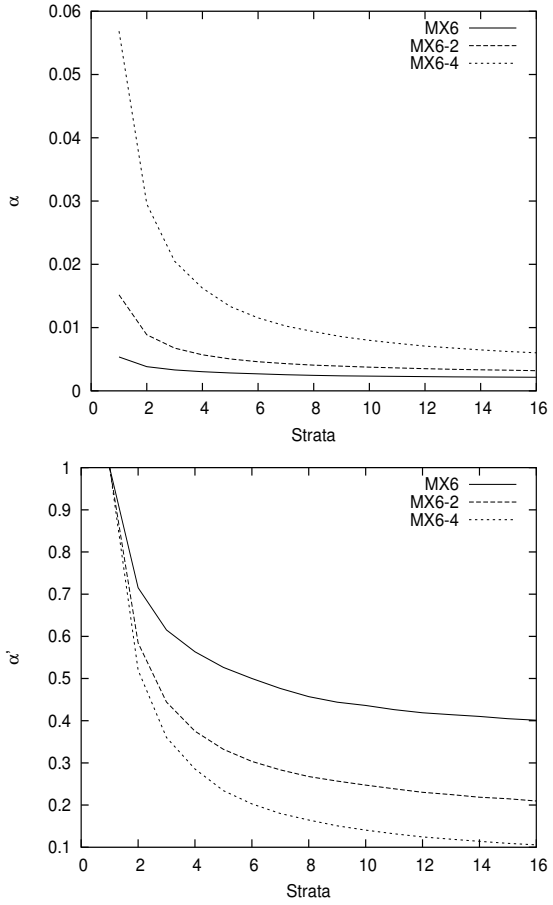
Figure 7.12 shows α values for MX6, MX6_2 and MX6_4. Clearly, α is strongly dependent on the number of instances in a dataset. As hypothesized above, time approximately behaves inverse proportional to the number of strata. To have a better insight in α, we compute α' as $\alpha'_s = \alpha_s/\alpha_1$, that is, the value of α for s strata over the value for one strata. Figure 7.12 also shows α'.

The evolution of α' can be approximated by a formula such as $\alpha' = a/s + b$, where s is the number of strata and b is a constant that needs to be adjusted to the problem at hand (from applying the formula for 1 stratum we know that $a = 1 - b$). In order to assign a value to b effectively developing a predictive model for α', we did some tests with several datasets of the MX6 family (with redundant bits and redundant instances) and performed a regression process. The results of these tests are in table 7.7 and show that b is mostly correlated to the number of instances in the dataset, and can be modeled as $b = c/\mathcal{D}+d$, applying regression again for c and d. These values, for an Athlon XP 2500+ with Linux and gcc 3.3 are 25.051 for c and 0.0270435 for d.

The model of α' is verified experimentally with two different datasets: MX11 and LED (using 2000 instances). LED was selected because it is similar to a real problem than the MX datasets due to the added noise. The comparison of the model and the empirical data can be seen in figure 7.13, which shows that the model is quite accurate.

With this α' model we can now deduce a formula to approximate the optimal number of iterations to maximize accuracy within a constant running time. The question is how many iterations using s strata (it_s) have the same run time as a base run time using one strata and

Figure 7.12: α (time per iteration) and α' (α relative to a single stratum) values for some datasets



it_1 iterations. it_s can be estimated by

$$it_s = \frac{it_1 \cdot s}{1 + b \cdot (s - 1)}, \tag{7.5}$$

setting $a = 1 - b$. This formula is quite interpretable: b is the overhead of the *GA* cycle. If it were 0, the speedup obtained would be optimal and we could do as many iterations as $it_1 \cdot s$ for s strata. This overhead, however, also depends on the number of strata showing that the stratification does affect not only the evaluation stage of the *GA* cycle but also the whole GA cycle.

Table 7.7: Values of constants a & b of the α' model for several datasets of the MX6 family. These values are highly dependent on the computer used (Athlon XP 2500+ with Linux and gcc 3.3 in this case)

Instances	Bits	a	b
64	7	0.646565	0.376474
128	7	0.773325	0.237257
128	8	0.748979	0.2611
256	7	0.857456	0.144431
256	8	0.855623	0.146535
256	9	0.841101	0.16176
512	10	0.90852	0.0910079
512	7	0.916831	0.0814565
512	8	0.920005	0.081788
512	9	0.919123	0.0540724
1024	10	0.95798	0.0398794
1024	11	0.953393	0.0450033
1024	7	0.959353	0.0365472
1024	8	0.956824	0.03408
2048	11	0.981321	0.017982
2048	8	0.985519	0.0177624

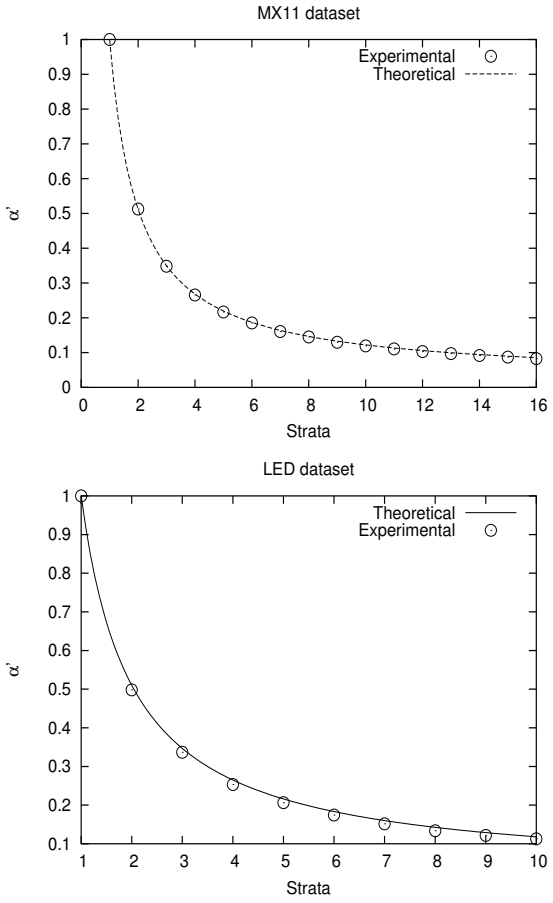
7.5 Testing ILAS in small datasets

This section puts into practice the *ILAS* windowing system in a large set of real datasets of small size. The objective of these tests is two-fold: on one hand we seek to determine if the models of the *ILAS* behavior presented in the previous section are valid for real datasets. On the other hand, it is time to validate in a rigorous way the good performance of *ILAS* reported in previous work (Bacardit & Garrell, 2003d; Bacardit & Garrell, 2003b).

To achieve both objectives, three strategies of the use of *ILAS* will be tested:

Constant Learning Steps (CLS) This strategy starts by setting a number of iterations of the system with only one strata. That is, without using windowing. This number of iterations is assigned by running the system until no further improvement of the training accuracy is achieved. Then, the configuration with two strata uses twice as many iterations, the configuration with three strata uses three times as many iterations, etc. The *learning steps* name comes from the *Michigan LCS* nomenclature, and means the number of input instances tested by the population in the learning process. In the context of GAssist, the meaning is approximately the same (the only difference is the last iteration of GAssist, which uses all the training set).

Figure 7.13: Verification of the $alpha'$ model with MX11 and LED datasets



Constant Time (CT) This strategy uses approximately the same run-time for all the configurations with different number of strata. The used run-time is the one achieved by the *constant learning steps* strategy with the non-windowing configuration. The run-time model described in the previous section and equation 7.5 are needed to use this strategy.

Constant Iterations (CI) In this strategy, all tested configurations with different number of strata will use the same number of iterations, the ones determined for the *Constant Learning Steps* strategy for the non-windowed configuration.

The methodology used for setting the initial number of iterations for the *constant learning steps* strategy is somewhat problematic, because it is quite probable that, in some of the datasets, it will mean generating solutions with over-learning. We think it is not a drawback for two reasons: (1) it will reflect even more the implicit generalization pressure introduced

Table 7.8: Number of iterations used for the non-windowed configuration of the *constant learning steps* strategy of *ILAS*

dataset	iterations	dataset	iterations	dataset	iterations
bal	500	h-s	250	thy	250
bpa	600	hep	250	vot	250
bre	1000	ion	450	wbcd	250
cmc	500	irs	250	wdbc	300
col	500	lab	250	wine	300
cr-a	500	lym	500	wpbc	300
gls	1500	pim	750	zoo	250
h-c	500	prt	750		
h-h	500	son	400		

by *ILAS* and (2) as the results of these experiments and the used statistical tests will show, using *ILAS* is beneficial in almost all datasets. This means that we will also have developed a systematic way of setting the number of iterations for *GAssist*. This issue is problematic in general for systems using the Pittsburgh model, because the number of iterations needed to generate competent solutions depends of several factors, like the size of the training set, the number of rules in the solution generated, etc.

After the results of each strategy are described, the best configurations for each of them will be compared.

7.5.1 The constant learning steps strategy

Table 7.10 shows the average results of this strategy. The full results on the selected experimentation framework containing 25 datasets are represented in table B.1 in appendix B. The non-windowed configuration plus four number of strata (2, 3, 4 and 5) were tested, making a total of 5 tested configurations for each dataset. For the sake of clarity, table 7.8 reproduces the number of iterations used for the non-windowed configuration on each dataset. Also, the parameters of the system were configured as described in table 7.9.

Looking at the average results of *ILAS* over all datasets we can see the general behaviour of this windowing schema: With the increase of the number of strata, the training accuracy and the number of rules decreases while the test accuracy increases until a certain point and then it too starts to decrease. From the analysis of the *ILAS* behavior in the previous section we can say that this decrease in test accuracy is due to the lack of representativity of the whole training set on each strata.

Table 7.9: Settings of GAssist for windowing experiments reported in section 7.5

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	400
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	auto
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Probability of Reinitialize (begin,end)	(0.02,0)
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

Table 7.10: Average results of the constant learning steps strategy tests of ILAS

#Strata	Training acc.	Test acc.	#rules	Run-time (s)
1	92.02±10.89	82.34±13.50	7.45±3.37	60.89±47.26
2	91.57±11.03	82.64±13.45	6.77±2.76	70.51±57.03
3	90.87±11.25	82.58±13.48	6.13±2.26	80.01±65.98
4	90.30±11.43	82.36±13.54	5.86±2.10	90.01±75.45
5	89.98±11.33	82.38±13.47	5.74±2.07	99.96±85.02

Table 7.11: Results of the t-tests comparing the 5 tests configurations of ILAS using the constant learning steps strategy, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	1 stratum	2 strata	3 strata	4 strata	5 strata	Total
1 stratum	-	0	1	1	1	3
2 strata	1	-	0	3	1	5
3 strata	1	0	-	0	1	2
4 strata	1	0	0	-	0	1
5 strata	1	0	0	1	-	2
Total	4	0	1	4	3	

Now it is time to determine which is the most suitable configuration of *ILAS* for the *constant learning steps* strategy. The most desirable option would be not to select a global best setting, but have a method to predict, for each dataset, which is the ideal number of strata. Actually, the aim of the model of stratification success described in previous section was exactly to be able to predict the most suitable number of strata for each dataset. However, using this model for real datasets (supposing that the number of rules that we obtain are equivalent to the number of niches in the dataset) has some problems:

- Not all rules cover the same number of examples. This problem is the smallest one, because we still can compute the stratification success probability, even if it not with a closed-form formula

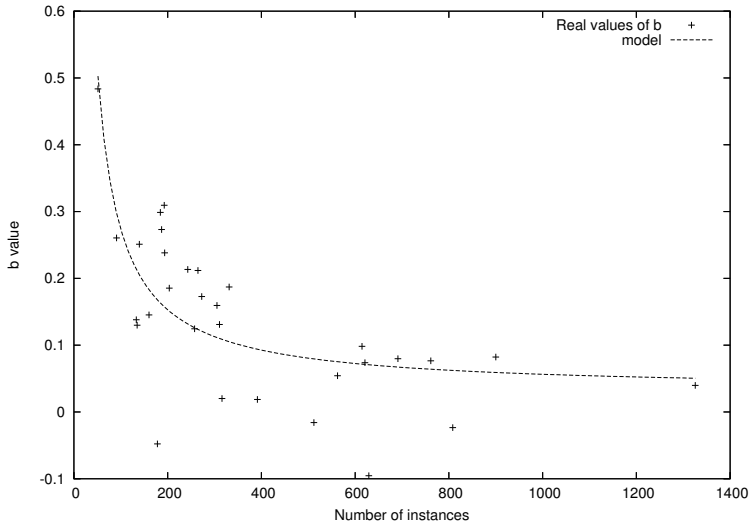
- The system might have split a niche into two rules, which would mean that we would create an over-pessimistic model.

- We cannot know if some rules cover examples that are simply noise, inconsistencies or outliers. This is the most difficult problem, and does not has a clear answer, because it is a basic part of the learning class: the generalization capacity.

This does not mean that the model is useless. It only means that some post-processing work is needed to successfully apply the model. Some heuristics have to be developed for merging and filtering the obtained rules, which is left as future work.

Meanwhile, with the help of some statistical tests, we seek to determine if there is some setting of *ILAS* can be considered good and robust enough to affirm that it can be used as the default configuration of *ILAS* in small datasets and *constant learning steps* strategy. The results of the statistical tests are summarized in table 7.11.

Figure 7.14: Overlapping the initial run-time model of ILAS with the experimental *b* values



From the results of the statistical tests it can be affirmed with high confidence that the configuration with 2 strata is the best one. It was the one achieving higher test accuracy average, it is the most robust one (it has never been significantly outperformed) and it is almost the top outperforming configuration. The specific comparison between the configurations with 2 and 3 strata indicates that their performance is never significantly different, but the slightly higher run-time of the configuration with 3 strata discards it as the best global candidate for the *constant learning steps* strategy.

7.5.2 The constant time strategy

In order to apply the constant time strategy the first step is determining the number of iterations needed for *ILAS* using 2, 3, 4 and 5 strata that have the same run-time as the non-windowed configuration, using the run-time model developed in previous section. However, as we can compute the value of the b parameter of the model for each dataset from the results of the *constant learning steps* strategy, we can check if the developed model is valid. Figure 7.14 overlaps the run-time model with the real b value for each dataset. It can be observed clearly in the figure that the model is not working correctly.

From figure 7.14 we can see several discrepancies between the model and the experimentation results, especially in the lower part of the figure, when the model is too pessimistic. If we look at which datasets have a low value of b, described in table 7.12, it can be observed that most of these datasets have a very low average of rules per individual. Therefore, we may

Table 7.12: Input information for the new run-time model for ILAS

Dataset	#Instances	Ave. #rules/indiv.	b
bal	562.500000	10.821058	0.054245
bpa	310.500000	10.233713	0.130907
bre	257.400000	12.242420	0.15417
cmc	1325.700000	8.734425	0.0397349
col	331.200000	16.507130	0.187061
cr-a	621.000000	8.123931	0.0736727
gls	192.600000	10.487166	0.309448
h-c	272.700000	10.834397	0.172864
hep	139.500000	8.403331	0.251041
h-h	264.600000	10.651623	0.211792
h-s	243.000000	10.624691	0.213375
ion	315.900000	5.802508	0.0201061
irs	135.000000	7.256767	0.129837
lab	51.300000	7.089468	0.483658
lym	133.200000	9.959550	0.13808
pim	691.200000	8.336094	0.0796481
prt	305.100000	19.265333	0.19428
son	187.200000	10.595582	0.273156
thy	193.500000	8.145829	0.238255
vot	391.500000	7.576427	0.0402058
wbcd	629.100000	5.076236	-0.0954714
wdbc	512.100000	5.631767	-0.0158894
wine	160.200000	6.582595	0.145283
wpbc	178.200000	4.696640	-0.0477674
zoo	90.900000	10.378047	0.267821

need to extend the model with another variable: the average number of rules per individual.

Therefore, from the information in table 7.12 a new model is extracted. This time the model takes the form defined in equation 7.6, where D is the number of instances in the training set and NR is the average number of rules per individual. The actual values for c, d and e are computed using linear regression [2] and are 30.3670, -0.9042 and 0.1257 respectively. The comparison of the new model with the experimental values of b is represented in figure 7.15.

$$b = c/D + d/NR + e \tag{7.6}$$

The new run-time model was used to determine the number of iterations used in the *constant time* strategy. Table 7.13 describes the average results of these tests. The full results

[2] Using the R statistical package to compute it

Figure 7.15: Overlapping the new run-time model of ILAS with the experimental *b* values

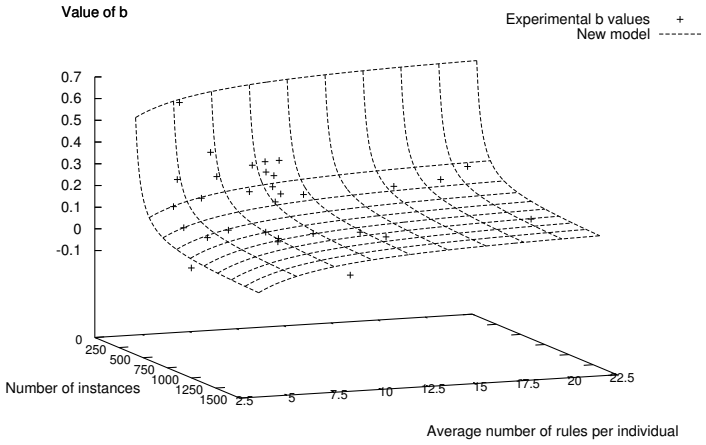


Table 7.13: Average results of the constant time strategy tests of ILAS

#Strata	Training acc.	Test acc.	#rules	Run-time (s)
1	92.02±10.89	82.34±13.50	7.45±3.37	60.89±47.26
2	91.47±11.04	82.53±13.38	6.78±2.68	61.71±49.42
3	90.69±11.37	82.50±13.42	6.15±2.25	62.64±50.50
4	90.06±11.53	82.51±13.34	5.87±2.11	63.46±51.37
5	89.56±11.64	82.23±13.47	5.72±2.03	64.64±52.23

are in table B.2 in appendix B. From the average run time it seems that the run-time model works relatively well, but in order to show exactly how well it is working, the relative run-time divergence between the configurations with 1 and 5 strata was computed for each dataset. Table 7.14 shows these divergences for each dataset. The average relative run-time divergence is 9.52%. This shows that the model is not perfect neither it is completely wrong. Only 9 of the 25 datasets had a divergence higher than 10%.

What is the best configuration for the constant-time strategy?. Again, statistical t-tests were computed, and its results described in table 7.15. This time configuration with 1 strata through configuration with 4 strata look equally robust. As strata 2 has higher average test accuracy, it is the best candidate as the default number of strata for the constant time strategy of *ILAS*.

Table 7.14: Relative divergence between the run time of ILAS configurations using 1 and 5 strata using *constant time* strategy

dataset	divergence	dataset	divergence	dataset	divergence
bal	8.72%	h-s	8.15%	thy	21.17%
bpa	0.73%	hep	7.57%	vot	8.73%
bre	15.38%	ion	18.94%	wbcd	7.36%
cmc	4.86%	irs	4.76%	wdbc	4.17%
col	8.90%	lab	7.65%	wine	5.60%
cr-a	5.17%	lym	12.06%	wpbc	7.24%
gls	23.19%	pim	0.38%	zoo	14.73%
h-c1	6.23%	prt	5.83%		
h-h	15.83%	son	14.74%		
Average	9.52±5.86				

Table 7.15: Results of the t-tests comparing the 5 tests configurations of ILAS with the constant time strategy, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	1 stratum	2 strata	3 strata	4 strata	5 strata	Total
1 stratum	-	1	1	1	1	4
2 strata	0	-	0	0	3	3
3 strata	0	0	-	0	1	1
4 strata	0	0	0	-	0	0
5 strata	0	0	0	0	-	0
Total	0	1	1	1	5	

Table 7.16: Average results of the constant iterations strategy tests of ILAS

#Strata	Training acc.	Test acc.	#rules	Run-time (s)
1	92.02±10.89	82.34±13.50	7.45±3.37	60.89±47.26
2	90.90±11.47	82.59±13.44	6.56±2.41	35.17±27.97
3	90.05±11.69	82.36±13.59	6.12±2.18	26.76±21.66
4	89.44±11.78	82.17±13.54	5.93±2.11	22.56±18.49
5	88.92±11.83	82.20±13.56	5.84±2.07	20.13±16.69

Table 7.17: Results of the t-tests comparing the 5 tests configurations of ILAS using the constant iterations strategy, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	1 stratum	2 strata	3 strata	4 strata	5 strata	Total
1 stratum	-	0	2	3	2	7
2 strata	1	-	1	2	1	5
3 strata	0	0	-	0	0	0
4 strata	0	0	0	-	0	0
5 strata	1	1	0	0	-	2
Total	2	1	3	5	3	

7.5.3 The constant iterations strategy

In this strategy, the tested numbers of strata in each dataset use the same number of iterations. The goal is two-fold: determine which is the number of strata that maximize the performance, and also which is the maximum run-time reduction that achieves the same performance as the non-windowed system.

The results of these tests are described in table B.3 in appendix B. The results are summarized in the average results in table 7.16. From these average results it seems that the the configurations that achieve the above stated goals are 2 strata and 3 strata, respectively. In order to verify these hints, statistical tests over these results were performed. Table 7.17 summarizes the results of the statistical tests, and shows how the configuration with 3 strata, although having an average test accuracy approximately equal to the non-windowed configuration, it is less robust. Therefore the two objectives stated above are achieved by the configuration with 2 strata.

Table 7.18: Comparison of the average results of the best strata configuration for the three ILAS strategies proposed. CLS = constant learning steps, CT = constant time, CI = constant iterations

Strategy	Training acc.	Test acc.	#rules	Run-time (s)
CLS	91.57±11.03	82.64±13.45	6.77±2.76	70.51±57.03
CT	91.47±11.04	82.53±13.38	6.78±2.68	61.71±49.42
CI	90.90±11.47	82.59±13.44	6.56±2.41	35.17±27.97

7.5.4 Comparing the results of the three strategies for ILAS

In this subsection the best configurations for each tested strategy are compared. Table 7.18 summarizes the performance of these configurations, all of them using 2 strata. To determine which is the best configuration we use, as usual, statistical tests. This time the tests indicated that there were no statistical differences between the performance of these configuration for any dataset. Therefore, it is reasonable to say that the *constant iterations* strategy is the best configuration of *ILAS* because it has similar performance to the other strategies but it uses less run-time.

7.6 Testing ILAS in large datasets

After verifying that the *ILAS* windowing scheme improves the overall performance of the system while reducing the run-time, it is time to check if it has good performance in the datasets where its use becomes critical: the large ones. The experimentation starts by testing the run-time model developed for the small datasets. Then, we focus on testing the performance of *ILAS* on the large datasets, defining an experimental methodology to tune the used number of strata.

7.6.1 Testing the ILAS run-time model on large datasets

The tests used to check the run-time model for *ILAS* developed for the small datasets will use the following procedure:

1. Running the system without using ILAS, to compute the reference number of iterations used in equation 7.5, is not feasible in general for large datasets for two factors: even if we only make short runs, it is not practical for run-time reasons (this is the motivation

for using a windowing process). For this reason, the reference number of iterations for the run-time model will already use *ILAS*. This means that we need a new formula for setting the number of iterations to achieve constant run-time with different degrees of stratification. This new formula is simply an extension of the previous one, and it is represented in equation 7.7, where it_{ref} is the iterations for the reference number of strata and s_{ref} is the reference number of strata.

$$it_s = \frac{it_{ref} \cdot s \cdot (1 + b \cdot (s_{ref} - 1))}{s_{ref} \cdot (1 + b \cdot (s - 1))} \tag{7.7}$$

2. A small number of runs (10) is made with constant time (150 seconds) using the selected reference number of strata. The reference number of strata is chosen in order to have a reasonable number of iterations in the selected run-time. The constant time of the runs is achieved simply by running the system until the pre-defined time limit is reached.

3. From there short runs, the average number of iterations and the average rules per individual are extracted.

4. With these two extracted measures and the size of the training set we can use the model for b and equation 7.7 to determine the iterations used for the other tested number of strata

The above defined steps are used for the *sick* dataset. The reference strata used are 5, that need 355 iterations to reach the selected run-time of 150 seconds. The other number of strata tested are 10, 15, 20 and 25. Table 7.20 contains the results of these tests. The settings of the system used are summarized in table 7.19. These results show that the model, although it is not perfect, it is relatively accurate. The maximum divergence in run-time is 8%.

The tests were repeated with another dataset (*nur*). This time the reference strata were 10, and the other tested settings used 20, 30, 40 and 50 strata. Table 7.21 has the results of these tests. In this case the run-time model does not work at all. The run-time of the configuration with 50 strata was almost 1/3 of the reference strata, which is not acceptable at all. What is the reason of such a big divergence from the model? if we compute the α (timer per iteration) values for the tested strata (listed in table 7.22), we see that α_{10} is more than double than α_{20}, which would mean that the overhead of the *GA* (b) was non-existent. However, if we look at the next values of α we can see how this observed tendency does not appear here. What is the cause of these contradictory observations?

We observe that the iterations in for the tests using 20 strata or higher are much faster that the iterations using 10 strata. The reason of this speed difference is the cache memory

Table 7.19: Settings of GAssist for windowing experiments reported in section 7.6

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	400
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1500
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	auto
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Probability of Reinitialize (begin,end)	(0.02,0)
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

Table 7.20: Results of ILAS on the sick dataset using the run-time model to achieve constant time

#Strata	Training acc.	Test acc.	#rules	Run-time (s)
5	97.59±1.91	97.35±1.91	6.42±0.62	147.73±17.35
10	98.59±0.60	98.23±0.87	6.26±0.64	138.07±14.09
15	98.52±0.60	98.08±0.83	6.41±0.83	137.34±9.65
20	98.33±0.85	98.02±0.99	6.50±0.85	139.16±9.44
25	98.25±0.65	97.87±0.91	6.59±0.84	135.83±8.75

Table 7.21: Results of ILAS on the nur dataset using the run-time model to achieve constant time

#Strata	Training acc.	Test acc.	#rules	Run-time (s)
10	94.63±0.97	94.55±1.08	16.27±6.27	152.00±12.51
20	94.21±1.22	94.10±1.37	12.82±2.79	104.66±7.54
30	93.57±1.28	93.47±1.40	11.69±2.45	81.37±6.51
40	93.25±1.35	93.14±1.51	10.88±2.23	66.91±4.92
50	93.12±1.22	93.02±1.48	10.35±2.20	57.98±4.01

Table 7.22: Alpha values (time per iteration) for the *nur* dataset and strata 10, 20, 30, 40, 50

#Strata	α
10	0.130133
20	0.0644874
30	0.0436561
40	0.0332035
50	0.0273878

of the computer. Our implementation of the data structure containing an instance uses 52 bytes for the *nur* problem, and each individual consumes an average of 539 bytes. Also, two full populations (parents and offspring) are maintained at the same time. This means that, given the 11665 instances of the dataset, the 400 individuals in the population and 10 strata, we are consuming 480KB of memory for storing the data used. Considering the code of the program running, and also the kernel of the operating system, it is very probable that from time to time some of this information is flushed from the 512KB of cache memory that the computer used in this experimentation has, slowing the program being run. With 20 strata we are using 432KB, with 30 strata the consume is 420KB. The consume is slowly getting far from the limit, and probably no data is being flushed from the cache memory.

What is the point of this observation? Probably we cannot make a general run-time model for ILAS in large datasets, because it depends on too many factors. If the dataset has real-valued attributes, the consume is much higher, therefore, domains with much less number of instances get already affected by this problem. Also, computers with only 256KB of cache memory are still very common nowadays, but in the high end of the market, computers with 1 or 2 MB of cache exist, ... It is not practical to create a model that we have to adjust too often. Therefore, the experiments conducted in next subsection will not use the run-time mode.

7.6.2 Testing the performance of ILAS in large datasets: tuning the number of strata for maximum performance

Looking at the results of *sick* and *nur* in the previous section we see some common behavior: the configuration having the best performance in the training set was also the best in the test set. What does this mean? By using a high number of strata, the implicit generalization pressure introduced by ILAS is so high that it is almost impossible to suffer from over-learning. Therefore, we only have to worry about finding the number of strata that maximizes the training accuracy.

In order to verify if this hypothesis can be generalized, we tested ILAS on all the datasets described in table 4.2 of chapter 4. We ran the system on each dataset with constant time, and testing, for each dataset, 5 different numbers of strata. Because the number of instances in the tested datasets range from 2300 to 100968, the same sets of strata cannot be used on all datasets. Therefore, each dataset will use a different set of tested strata. The criteria used to select these sets is the same used in previous subsection, strata that use a reasonable (at least 250) number of iterations in the predefined time.

Two different time limits are used: 150 and 300 seconds. The aim of these two limits is testing two degrees of "reasonable" run-time (although nowadays there are several machine learning algorithms that run in very short time). If we suppose we are using this learning system in a real-life environment, spending 5 minutes to learn a knowledge base is still quite a reasonable duration. Table 7.23 contains the results for time limit 150 and table 7.24 contains the results for the runs with 300 seconds of duration.

Table 7.23: Results of ILAS on large-size datasets with time-limit 150

Dataset	#Strata	Training acc.	Test acc.	#rules
adu	75	85.28±0.23	85.12±0.59	10.63±1.93
	100	85.25±0.25	85.11±0.57	10.28±2.10
	200	85.04±0.31	84.94±0.55	9.97±2.02
	300	84.56±0.69	84.44±0.79	10.19±2.14
	400	84.18±0.83	84.09±0.93	10.20±2.03
c-4	50	69.77±2.37	69.64±2.31	11.65±4.86
	75	70.41±2.42	70.22±2.34	13.49±5.96
	100	70.21±2.77	70.09±2.71	12.58±5.57
	125	70.03±2.84	69.94±2.78	11.96±5.53
	150	70.01±2.70	69.90±2.63	10.69±4.30

Table 7.23: Results of ILAS on large-size datasets with time-limit 150

Dataset	#Strata	Training acc.	Test acc.	#rules
fars	250	76.78±0.55	76.71±0.56	13.44±2.20
	500	76.48±0.50	76.48±0.50	13.22±2.31
	750	76.21±0.41	76.21±0.45	12.74±2.18
	1000	76.07±0.56	76.05±0.54	13.06±2.28
	1250	75.77±0.30	75.75±0.27	13.12±2.22
hyp	10	94.53±0.49	94.25±0.64	6.95±0.95
	15	94.42±0.49	94.17±0.65	7.01±1.14
	20	94.25±0.54	94.03±0.67	7.19±1.17
	25	94.27±0.40	94.05±0.62	7.05±1.06
	30	94.20±0.42	94.04±0.57	6.98±0.92
krkp	5	96.71±1.64	96.68±1.82	7.25±0.46
	10	96.59±1.65	96.51±1.83	7.23±0.56
	15	96.32±1.65	96.18±1.80	7.17±0.49
	20	96.07±1.63	95.92±1.81	7.10±0.32
	25	96.45±1.58	96.34±1.78	7.27±0.68
mush	5	99.80±0.35	99.79±0.38	4.78±0.85
	10	99.88±0.28	99.87±0.31	4.87±0.90
	15	99.88±0.26	99.86±0.31	4.92±0.80
	20	99.86±0.28	99.84±0.31	4.99±0.77
	25	99.83±0.31	99.81±0.33	4.87±0.73
nur	10	94.69±0.90	94.54±1.09	16.71±6.59
	20	94.47±1.01	94.41±1.20	13.67±3.51
	30	94.16±0.97	94.06±1.07	12.03±2.35
	40	93.85±1.13	93.75±1.20	11.45±2.06
	50	93.70±1.04	93.60±1.16	11.28±2.06
pen	25	68.36±2.60	68.10±2.89	12.63±2.07
	50	69.48±2.57	69.24±2.85	11.81±1.70
	75	68.91±2.33	68.68±2.58	11.76±2.09
	100	68.21±2.65	68.01±2.73	10.93±1.79
	125	67.00±2.15	66.94±2.58	10.86±1.64
sat	10	77.64±1.15	77.40±1.71	8.22±1.75
	20	79.42±0.70	78.95±1.42	8.78±1.80
	30	79.67±0.60	79.28±1.27	9.05±1.84
	40	79.62±0.63	79.42±1.35	8.27±1.45
	50	79.52±0.58	79.18±1.27	8.13±1.61

Table 7.23: Results of ILAS on large-size datasets with time-limit 150

Dataset	#Strata	Training acc.	Test acc.	#rules
seg	5	89.02±1.59	88.11±2.49	8.83±1.38
	10	89.86±1.11	89.04±2.25	8.41±1.13
	15	89.56±1.36	88.87±2.32	7.95±1.13
	20	89.48±1.25	88.92±2.24	7.72±0.88
	25	89.38±1.33	88.91±2.17	7.67±0.91
sick	5	97.69±1.82	97.31±1.79	6.52±0.75
	10	98.59±0.65	98.21±0.88	6.33±0.64
	15	98.57±0.33	98.18±0.81	6.29±0.56
	20	98.46±0.46	98.09±0.80	6.45±0.77
	25	98.33±0.29	97.98±0.76	6.43±0.79
spl	5	91.24±2.04	90.37±2.61	9.31±2.20
	10	92.54±1.36	91.59±1.96	9.72±2.26
	15	92.25±1.63	91.50±2.01	8.31±1.38
	20	91.66±1.61	90.88±2.10	8.21±1.36
	25	90.71±2.60	90.24±2.84	8.07±1.11
wav	10	76.85±0.78	75.16±2.12	9.32±1.75
	20	77.50±0.61	75.73±2.16	9.83±1.76
	30	77.16±0.65	75.47±2.17	9.40±1.64
	40	76.92±0.61	75.14±2.23	8.91±1.48
	55	76.59±0.69	75.03±2.32	8.93±1.78

Table 7.24: Results of ILAS on large-size datasets with time-limit 300

Dataset	#Strata	Training acc.	Test acc.	#rules
adu	75	85.23±0.24	85.09±0.56	10.79±2.18
	100	85.27±0.24	85.10±0.56	10.63±2.20
	200	84.99±0.33	84.85±0.60	10.37±2.04
	300	84.71±0.53	84.59±0.72	10.11±2.22
	400	84.32±0.72	84.26±0.88	10.25±2.20
c-4	50	71.92±2.39	71.69±2.32	21.88±8.80
	75	71.75±2.80	71.54±2.71	19.27±8.74
	100	71.46±2.97	71.24±2.85	17.72±8.24
	125	71.34±2.80	71.14±2.74	15.19±7.28
	150	71.21±2.61	71.05±2.57	13.31±6.05

Table 7.24: Results of ILAS on large-size datasets with time-limit 300

Dataset	#Strata	Training acc.	Test acc.	#rules
fars	250	76.83±0.57	76.80±0.58	13.58±2.43
	500	76.36±0.44	76.35±0.49	12.58±2.17
	750	76.27±0.49	76.20±0.44	13.02±2.34
	1000	75.86±0.49	75.84±0.47	13.02±2.65
	1250	75.83±0.35	75.86±0.35	13.50±2.57
hyp	10	94.77±0.50	94.47±0.77	6.85±0.85
	15	94.62±0.54	94.39±0.68	7.01±0.98
	20	94.49±0.56	94.21±0.73	7.00±0.95
	25	94.42±0.50	94.24±0.72	7.05±1.02
	30	94.25±0.37	93.99±0.50	7.08±0.95
krkp	5	97.36±1.44	97.27±1.67	7.40±0.62
	10	97.74±1.14	97.58±1.32	7.46±0.58
	15	97.55±1.18	97.44±1.29	7.24±0.65
	20	97.46±1.21	97.35±1.38	7.20±0.57
	25	97.79±0.62	97.67±0.80	7.19±0.53
mush	5	99.90±0.26	99.90±0.28	4.83±0.89
	10	99.94±0.21	99.93±0.23	4.99±0.85
	15	99.96±0.14	99.95±0.19	4.96±0.78
	20	99.94±0.18	99.92±0.24	5.03±0.74
	25	99.95±0.10	99.94±0.13	4.97±0.67
nur	10	95.39±0.93	95.23±1.10	19.80±7.21
	20	95.11±0.92	94.97±1.03	13.83±2.92
	30	94.75±0.85	94.63±1.01	12.83±2.51
	40	94.49±0.82	94.38±0.98	11.96±1.90
	50	94.38±0.77	94.33±0.97	11.51±1.57
pen	25	72.18±2.68	71.93±2.96	12.68±1.75
	50	72.02±2.37	71.65±2.51	11.89±1.82
	75	70.96±2.45	70.89±2.73	11.37±1.74
	100	69.94±2.41	69.64±2.67	11.63±1.75
	125	68.92±2.15	68.86±2.27	10.81±1.63
sat	10	79.62±0.64	79.16±1.46	9.02±2.19
	20	80.34±0.53	79.85±1.35	8.77±1.98
	30	80.45±0.62	80.00±1.30	8.25±1.56
	40	80.19±0.50	79.72±1.32	7.93±1.43
	50	80.01±0.53	79.63±1.34	7.71±1.59

Table 7.24: Results of ILAS on large-size datasets with time-limit 300

Dataset	#Strata	Training acc.	Test acc.	#rules
seg	5	90.68±1.16	89.98±2.06	8.93±1.25
	10	90.89±1.08	90.25±2.11	8.09±1.06
	15	90.78±1.07	90.15±2.10	7.95±0.88
	20	90.46±1.12	90.10±2.11	7.90±1.09
	25	90.14±0.98	89.69±1.95	7.67±0.79
sick	5	98.65±0.77	98.23±0.95	6.24±0.49
	10	98.75±0.23	98.41±0.70	6.33±0.66
	15	98.67±0.23	98.29±0.72	6.30±0.67
	20	98.54±0.24	98.18±0.65	6.33±0.70
	25	98.41±0.23	98.06±0.70	6.51±0.81
spl	5	93.58±2.53	92.46±2.79	11.67±4.23
	10	93.58±1.09	92.45±1.70	8.95±1.71
	15	93.01±1.05	92.47±1.86	8.47±1.39
	20	92.06±2.15	91.49±2.65	8.19±1.23
	25	92.03±1.49	91.19±2.14	8.27±1.31
wav	10	78.28±0.60	76.01±1.97	10.64±1.94
	20	78.15±0.60	76.27±1.95	9.34±1.52
	30	77.80±0.63	75.88±2.03	9.14±1.57
	40	77.39±0.60	75.38±2.26	9.15±1.67
	50	76.95±0.62	75.26±2.08	8.97±1.57

From the results on these 13 datasets, in 12 of them the above stated hypothesis is correct (for both types of experiments done). Even, in the dataset where the best training accuracy and the best test accuracy did not belong to the same number of strata, the accuracy difference from the best method in test is minimal.

Moreover, does *ILAS* benefit from running for 300 seconds compared to the 150 seconds runs? Statistical tests were made to compare the performance of the best configuration for each dataset in the two type of tests. The paired t-tests determined that the runs with 300 seconds were significantly better than the runs with 150 seconds in all but *adu* and *fars* datasets, using a confidence level of 99%. If we look at the results, the average accuracy difference in datasets such as *sick* seems small (only 0.2%), but looking at the smaller deviation we can see why this small difference became significant: the system is much more stable if it runs for longer time.

7.7 Discussion and further work

In this chapter we have seen the *ILAS* windowing scheme used in different scenarios. In all of these scenarios the performance of the system benefits from the use of *ILAS* in a significant way.

In small datasets we have seen three different strategies for the use of *ILAS*. These strategies represent three degrees of balance between performance (accuracy) and run-time. The first strategy (*constant learning steps*) maximizes performance with a small sacrifice of run-time. The second strategy (*constant time* is a middle point: determining the maximum performance increase achievable using *ILAS* with the same run-time as the non-windowed system. The third strategy (*constant iterations*) seeks the maximum run-time reduction having a competent performance compared to the non-windowed system.

These three strategies were tested over 25 datasets. The results of these tests and the statistical tests applied over these results showed in a significant way which is the best (in both performance and robustness) configuration for each kind of strategy. Therefore, we have a systematic and versatile methodology to tune the *ILAS* windowing system for further small datasets. Also, we can affirm with high confidence that *ILAS* is useful in almost all configurations, even if run-time reduction is not an important issue, because the tests have showed how *ILAS*, with its implicit generalization pressure, can reduce the sensitivity to over-learning of GAssist.

In large datasets *ILAS* is also useful in two ways: It can reduce in a very significant way the run-time of the system (even using thousands of strata, like in *fars*) and it can also help the system to learn better. In a Pittsburgh system with a global fitness function and given these large datasets with thousands, hundreds of thousands or even millions of instances, the contribution of classifying correctly each training instances becomes so small that it makes the learning process even more difficult. If we can partition the training set in small chunks (the strata), we help the system process correctly the training set by achieving a fitness landscape that allows the selection algorithm to choose properly between individuals.

Like in the small datasets, we also have a deterministic process to tune the number of strata that maximizes the performance for each dataset. This process worked correctly for 12 of the 13 datasets used in the experimentation. Also, in most datasets we determined in a significant way that the system needs more than 150 seconds (using modern hardware) to learn properly. This deterministic process needs some improving. Right now we have determined that it is correct by performing a full test in all candidate configurations. The ideal situation would be to determine the ideal number of strata for each dataset using only short runs. Therefore, this tuning process needs further development.

Thus, we have shown experimentally that the performance of the system is good in many different scenarios. However, we cannot say the same of the two models we developed about the behavior of the system.

The goal of the first model was to predict the maximum stratification degree we can use before some of the strata lose the representativity of the whole training set. This model was quite good for synthetic problems, but when we jumped to real problems, the situation was different. With the current model, and making the supposition that the rules we obtain are more or less equivalent to the niches existing in the training set, we cannot predict accurately the number of strata that maximizes the test performance of the system. Maybe this extrapolation from training to test is impossible to assume. But there is another path worth of exploring. Probably we need to filter the obtained rule set in order to predict more accurately the number (and coverage) of the niches existing in the training set. With some post-processing of this kind, maybe we can obtain an enough accurate model that outperform the deterministic tuning process of *ILAS* that we have experimentally verified that is already quite robust.

The other model was about the run-time of the system. With the synthetic datasets that were used to develop the model, we could predict the relative run-time (relative to the non-windowed system) of the system knowing only the size of the dataset. The tests with real problems showed that the model needs another variable: the average number of rules per individual during the learning process. With the extended model with two variables we can predict the relative run-time for small datasets more accurately (although it is not perfect). However, the model cannot be extrapolated to large datasets, because of the experimental limitations we have with the hardware used. The cache memory, while being beneficial for the run-time of any program in most situations, makes impossible the development of a general model of run-time for large datasets, because the parameters of the model need to be tuned almost for every datasets, eliminating totally the predictive capacity of the model.

The reader may wonder why is it worth using a run-time model, if we can simply stop the learning process when the time limit is reached. The point is that the mechanism used is unfair because the number of iterations achieved in this fixed time can change if, for random reasons, the average number of rules per individual (the other factor, beside the size of the training set that controls the cost of the fitness function) differs from run to run. Therefore, some of the runs cannot learn enough if the contain, by chance, large individuals. An alternative is determining an approximate number of iterations using the pre-defined fixed time using only few runs, and then perform the full test with fixed number of iterations, but this has some extra cost. If we could have a reliable run-time model, this extra cost would be much lower.

7.8 Summary of the chapter

In this chapter we have described the work done on applying windowing techniques to a Pittsburgh approach genetic-based machine learning system. The objective of these methods is to reduce the cost of fitness computations by using only a subset of the training examples to evaluate each individual and, therefore, reducing the total computational cost of the system.

Most of the chapter has been focused on an specific windowing technique, called *ILAS* (incremental learning with alternating strata). This technique divides the training set into several non-overlapped subsets of equal class distribution as the whole training set, therefore, strata. Then, each iteration uses a different strata, using a round-robin policy.

The chapter started by describing the historical process and motivation for the development of *ILAS*, by showing the prior attempts at windowing schemes and some previous results. The chapter continued by describing the work done on analyzing the behavior of *ILAS*, which led to the development of two models. One about the maximum number of strata that can be used without degrading the performance of the system. By this we mean computing the probability that the created strata are still enough representative of the whole training set. The other one about the run-time of *ILAS*. Both models were developed using synthetic (and thus, predictable) datasets.

When these two models were put into practice using real datasets, the experimentation showed that the stratification representativity model needs to be refined if it is to predict the number of strata that give maximum test accuracy. As further work, some ideas of how to do this refinement were proposed. The experimentation with real datasets showed that the run-time model needed to be expanded. The expanded model was more accurate in small datasets, but it was not usable in large datasets, because of physical limitations of the experimentation framework.

Independently of these two models, *ILAS* showed in the experimentation on real datasets that it can improve the performance of the Pittsburgh model of GBML in more than one way. The tests showed how *ILAS* can be used successfully in several different scenarios, showing its versatility.

Chapter 8

Bloat control and generalization pressure methods

Bloat control and generalization pressure are very important issues in the design of Pittsburgh GBML systems, in order to achieve simple and accurate solutions in a reasonable time. The bloat control deals with a problem, identified as bloat effect, *related to the growth without limit of the size of the individuals. The same techniques used to control bloat if properly adjusted and combined with other techniques can be helpful in the introduction of generalization pressure into the system, evolving more accurate but compact solutions potentially having better test accuracy. A side effect of applying this pressure towards short individuals is a run-time reduction, always desirable in this context.*

Thus, the chapter will present several mechanisms intended to control the bloat effect and apply generalization pressure. Some of them can be combined, some of them not. An analysis of each mechanism, reporting how they affect the general dynamics of the system, and how can they be used properly will be presented. Finally, a comparison of the non-combinable techniques with an external reference will be presented.

The chapter is structured as follows: First, section 8.1 will show a larger introduction to the chapter. Next, section 8.2 will describe briefly some related work, followed by section 8.3 containing a short description of how Bloat effect *affects Pittsburgh GBML systems, and also some guidelines about how should the measures to solve the* Bloat *effect be defined. Section 8.4 will provide the basic mechanism used to control the bloat effect: a rule deletion operator. The operator will be defined and its behavior studied. After describing the bloat control method, section 8.5 will contain the generalization-pressure methods studied. Like in the previous section its behavior will be analyzed, and some additional mechanisms that guarantee that the learning process is performed properly will be introduced. Section 8.6 will show the*

experimentation performed on a large set of domains to analyze in depth the generalization pressure methods presented in the previous section, comparing them to an alternative method. This comparison, together with the analysis of the bloat control method will lead to some discussion and further work, presented in section 8.7. Finally, section 8.8 will summarize the chapter.

8.1 Introduction

One of the most important problems of variable-length Pittsburgh GBML systems is the control of the bloat effect. As described in section 3.6, the bloat effect is characterized by a growth without control of the size of the individuals, affecting in general any variable-length representation used in evolutionary computation. This chapter describes the contributions made in this thesis to control this effect.

Nevertheless, as the title of this chapter suggests, we want to go one step further. The same techniques that can control successfully the bloat effect, if adjusted properly, can also introduce some extra generalization pressure into the behavior of the system. This issue is very important considering the base model we have taken as initial point (*GABIL*) which has a fitness function that only considers the accuracy of the whole rule set over the training examples, without any explicit mechanism related to the complexity of the rule-set. Given this fitness function, the easiest way to increase it is to maximize the probability of correctly classifying the training examples, which is achieved by increasing the size of the individuals. This fact produces solutions that are bigger than necessary, contradicting the *Occam's razor* principle which says that "the simplest explanation of the observed phenomena is most likely to be the correct one". A probable consequence of the "over-complexity" is an over-fitting of the solutions created which can lead to a decrease of the generalization capacity. The following techniques will be described and studied in the chapter:

Rule pruning operator (Bacardit & Garrell, 2002c) This operator removes useless rules from the individuals. It is applied after the fitness computation, when it is known which rules have never been used (thus useless). Some constraints control the disruptive potential of the operator

Hierarchical selection operator (Bacardit & Garrell, 2002c) This operator is actually a comparison function integrated into the tournament selection, to decide the outcome of each tournament. In order to decide which individual is better, it uses a double-step comparison, selecting the smallest individual if the accuracies of the individuals involved in the

tournament are *similar*.

MDL-based fitness function (Bacardit & Garrell, 2003a) This is a fitness function based on the *minimum description length* (MDL) principle (Rissanen, 1978), this is a metric that combines the accuracy and the complexity of a solution in a smart way to obtain a fitness function aware of both factors. The complexity of a solution is dependent on the knowledge representation used, and it allows us not only to promote smaller solutions, but better ones, depending on how the rules are defined.

Fitness penalty for short individuals The two above individuals are helpful in the promotion of compact individuals, but they are also dangerous if applied without control, because they may create an irreparable loss of diversity and information in the population. A simple penalty function is aggregated with the fitness function to avoid this situation.

The first technique of this list will be studied in section 8.4, the other three in section 8.5.

8.2 Related work

The *MDL* principle has been applied as a part of modeling tasks in many different fields. For example, handwriting recognition and robotic arms (Gao, Li, & Vitányi, 2000). , determining the topology of an Artificial Neural Network (Lehtokangas, Saarinen, Huuhtanen, & Kaski, 1996) applied to time series prediction or magnetic recording channels (Kavcic & Srinivasan, 2001). The principle has also been widely applied for classification tasks. Some examples are the creation of decision trees by means of Inductive Learning (Quinlan & Rivest, 1989; Forsyth, Clarke, & Wright, 1994) , Genetic Programming (Iba, de Garis, & Sato, 1994) , Constructive Induction (Pfahringer, 1994), Bayesian Networks creation (Wong, Lam, & Leung, 1999) or, probably, the best known case: c4.5rules (Quinlan, 1993; Quinlan, 1995), where the *MDL* principle is used to select the best subset of rules derived from a c4.5 induced decision tree.

Section 3.6 already described how the bloat effect is controlled in different paradigms of evolutionary computation. The rule-deletion operator used here is probably closer to the one used in *SAMUEL* (Grefenstette, 1991) than the one used in *GIL* (Janikow, 1991). There were other examples reported (Aguirre, González, & Pérez, 2002; Luke & Panait, 2002) of techniques similar to the hierarchical selection operator. The main difference is the criteria used to jump from the primary decision factor of the tournament to the other factors. As stated in the previous paragraph, the *MDL* principle has been used in the genetic programming field, although using a different formulation from the one used here, due to knowledge representation differences. The closest technique in general to the ones studied here is a bloat control

technique for Pittsburgh GBML systems based on multi-objective optimization called *MOLCS* (Llorà, Goldberg, Traus, & Bernadó, 2002) [1]. In section 8.6 the methods studied in this chapter will be compared to this MOO-based approach

8.3 The bloat effect: why it happens and how we have to deal with it

In this section we will do a brief and illustrative introduction about how and why the bloat effect affects Pittsburgh GBML. We will also show that fixing this problem is not a simple task, showing how bad ways to fix this problem can collapse the learning process.

8.3.1 What form does the bloat effect take?

Usually the bloat effect is defined as the growth of the individuals length without control, and it is a phenomenon that can affect in general all variable-length representations. In Pittsburgh LCS this effect takes the form of an exponential-rate growing of the number of rules of the individuals. This effect can be illustrated by the first 15 iterations in figure 8.1, which represents the evolution of the average individual size for the *MX11* problem. If we did not apply any measure to control this, the program would crash for lack of memory shortly after.

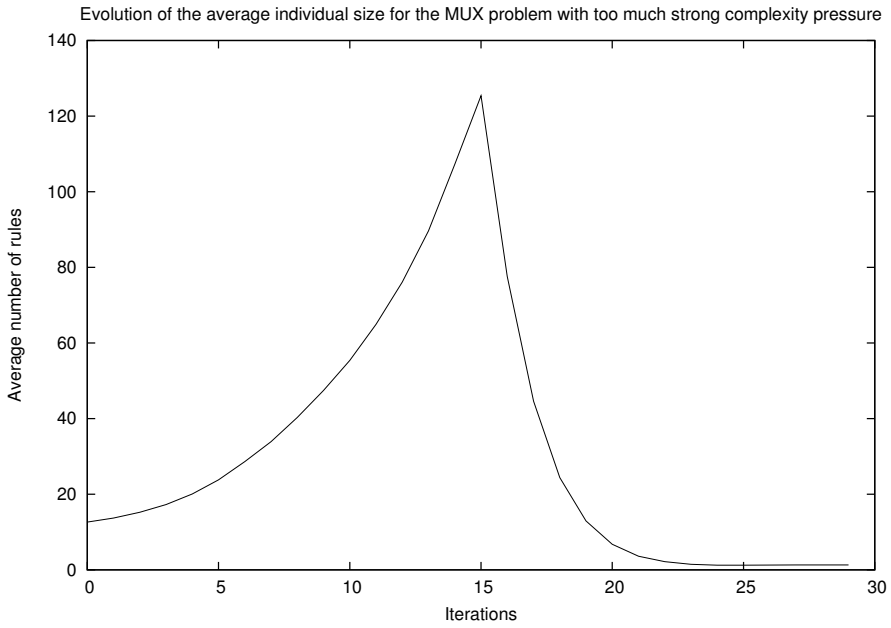
8.3.2 Why do we have bloat effect?

The reason of the bloat effect is well explained in (Langdon, 1997). Its cause is the use of a fitness function which only takes into account the validity of the solution (accuracy in our case). Having a variable-length representation means that it possible to have several individuals with the same fitness value, and there will be more long representations of a given solution (fitness value) that short ones. So, when the exploration finds new solutions, it is more probable that these solutions will be long than short.

The interpretation of this idea in *LCS* is that, it is more probable to classify correctly more training examples with an individual with a lot of rules that with a short individual. Is this long individual a good solution? Probably no, as this individual is *memorizing* the training examples instead of *learning* them. This shows a side effect of bloat in *LCS*: the generated solutions will probably lack generalization, and its test accuracy will probably be poor.

[1] Specifically, we are using the *MOLCS-GA* version of *MOLCS*

Figure 8.1: Illustration of the bloat effect and how a badly designed bloat control method can destroy the population



8.3.3 How can we solve the bloat effect?

It is obvious that we need to add to the system some bias towards good but simple solutions, but will any intervention in this sense work? The answer is no. If we introduce too much pressure towards finding simple solutions, we are in danger of collapsing the population into individuals of only one rule, which cannot generate longer individuals anymore. With these kind of individuals we can only classify the majority class. Again in figure 8.1 we can see an example of too much pressure for the MX11 dataset, which is activated just after 15 iterations. With only a few iterations, a population of an average of more than 120 rules per individual is reduced to individuals containing only one rule. The bloat control method that created this situation is the MDL-based fitness function studied in this chapter, but using a bad parametrization (InitialRateOfComplexity=0.5).

Figure 8.2: Code of the rule deletion operator

Rule deletion operator
Input : $Individual, RuleActivations, minThreshold$
$numRules = Individual.numRules$
$RulesToDelete = \emptyset$
For $i = 1$ to $numRules$
  If $RuleActivations[i] = 0$
    Add i to $RulesToDelete$
  EndIf
EndForEach
$selectedRules = |RulesToDelete|$
If $numRules - selectedRules < minThreshold$
  $selectedRules = numRules - minThreshold$
  While $|RulesToDelete| > selectedRules$
    pos = random$[1, |RulesToDelete|]$
    Remove position pos from $RulesToDelete$
  EndWhile
EndIf
Remove rules in $RulesToDelete$ from $Individual$
Output : $Individual$

8.4 Controlling the bloat effect

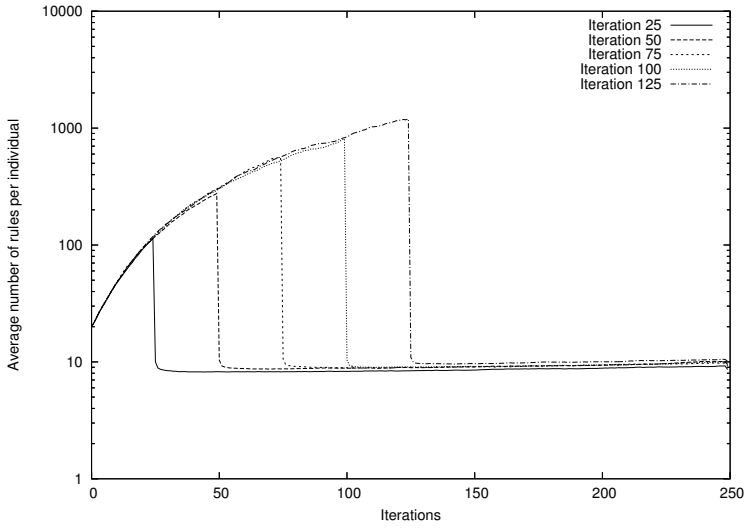
The mechanism studied to control the bloat effect is a rule deletion operator. This operator is applied after every fitness computation, removing the rules of the individual that have not been activated with any input example. The behavior of the operator is controlled by two constrains:

- The process is only activated after a predefined number of iterations, to prevent an irreversible diversity loss.

- The number of rules of an individual never goes below a threshold. If the rules of an individual that are selected for deletion will lower the total number of rules below this threshold, only the extra rules (over the threshold) are deleted. How do we decide which rules to delete? In order to introduce as little bias as possible, the subset of rules that are deleted is randomly selected.

Algorithmically, the operator is represented in figure 8.2.

Figure 8.3: Evolution of the number of rules for the *pim* dataset depending on the activation iteration of the rule pruning operator. Log scale on the y axis

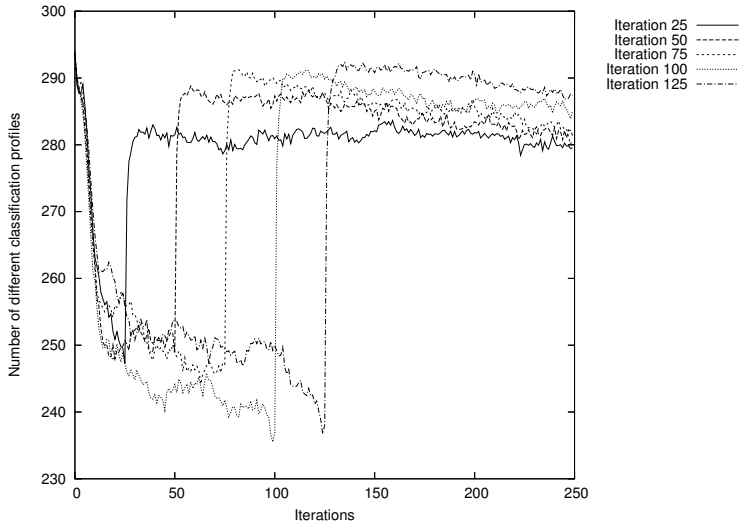


8.4.1 Tuning the iteration of activation of the operator

What is the effect of this operator in the population? Figure 8.3 shows the expected effect over the number of rules in the individuals, using 5 starting iterations: 25, 50, 75, 100 and 125 for the *pim* dataset. The figure shows how the average number of rules in the population can increase up to more than 1200 rules if it is not controlled. This means a memory consumption for this dataset and a population size of 300 of approx. 70MB. Considering that this dataset is relatively small, it is unacceptable to consume that much memory. Also, the run that started pruning individuals at iteration 125 had a run time three times larger than the one starting at iteration 25.

Thus, it seems reasonable to activate the rule deletion operator early in the learning process. Nevertheless, is there any additional motivation for this early activation? The answer again is yes, and the cause is that the bloat effect produces a loss of diversity in the population. This effect is not particular in this kind of system, it also affects genetic programming, where there is the widespread interpretation that individual size growth occurs to protect the individuals from the destructive effects of the recombination operators such as crossover (Soule & Foster, 1998). When the size of the trees grows, most of the code in the individuals is useless (like in our case). In the GP literature this useless code is defined as *neutral code* or *introns*. If most of the individual code are introns, the crossover operator will have more chances of manipulating

Figure 8.4: Evolution of the number of different classification profiles in the population for the *pim* dataset depending on the activation iteration of the rule pruning operator



an intron than manipulating useful code, which will prevent the destruction of the useful parts of the individual. However, as the code grows, the chances of improving the fitness of the individuals by recombination also decrease for the same reason, thus producing a diversity loss in the population.

The problem with diversity in Pittsburgh GBML is easy to illustrate by using the concept of *classification profile*. Given an individual Ind, a training set $\mathcal{D}$ formed by instances $(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n)$ and a prediction function $pred(Ind, Ins)$ that given an Individual and an input instance it returns the class predicted by the individual for the input instance, we can define the classification profile CF as a vector of size n (the size of the training set) defined as:

$$CF(Ind, \mathcal{D}) = (pred(Ind, \mathcal{D}_1), pred(Ind, \mathcal{D}_2), \ldots, pred(Ind, \mathcal{D}_n)) \tag{8.1}$$

This vector defines the behavior of an individual. If we count the number of different vectors of this kind that can be found in the population, we have an effective diversity measure of the population. Figure 8.4 shows the evolution of the number of different profiles through the iterations for the same tests with the *pim* dataset shown if figure 8.3. The figure shows how the number of different profiles descends gradually until the rule deletion operator is activated. Then it suffers a drastic increase and then it starts a more slower descent while the population converges towards the good solutions.

Looking at the y scale of the plot in figure 8.4 it can be observed that this diversity loss

is not drastic, the minimum achieved number of profiles is 235 (the maximum achievable is 300, the population size). However, some other datasets can have a much lower number of different profiles, and therefore be in real danger of a critic diversity loss.

Considering that for two reasons, run-time and diversity, it seems suitable to start pruning individuals in early iterations. Our previous tests indicated that starting pruning at iteration 5 is safe enough to guarantee that there are enough alive rules in the population, but it is very early, so the impact to the population diversity and the run time is very minor.

8.4.2 Tuning the lower threshold of activation of the operator

There is another parameter that needs tuning, the parameter $minThreshold$. This parameter disables the operator if the number of rules in the individual becomes less or equal to this parameter, even if there still are dead rules in the individual. Why it would be useful to leave some dead rules in the individual? The reason recalls again the arguments described in the previous subsection about the *introns*. Some small quantity of introns can be beneficial for the individuals because they may protect them from the destructive effects of crossover. The question is to define what is small quantity. The answer is difficult to determine. As a rule of thumb, in the experimentation of this chapter this threshold will be set to the number of alive rules of the final solution + 3. This means that some short runs are required to tune the system. The automatic setting of this threshold is left as further work.

This tuning of $minThreshold$ has a consequence: the generalization pressure methods presented in next section will need to be aware of the existence of useless rules, and ignore them. If not, these useless rules tend to disappear because they do not have positive contribution to the fitness of the individual. There is another consequence: what happens with these useless rules in the test stage of the system? It is obvious that these rules are useless in the training set, but it cannot be guaranteed the same for the test set. In order to avoid *random* behavior of the generated solutions in the test stage because of these dead rules, $minTheshold$ will be lowered to value 1 for the last iteration of the learning process. In this way we want to guarantee that the behaviour showed by the best individual of the population in the test stage is a representation of what it has learned during the training process.

8.5 Applying extra generalization pressure

In this section we describe our contributions in methods designed explicitly to apply generalization pressure to a Pittsburgh GBML system. Two alternative methods are presented, the first is a very simple approach: the hierarchical selection. The second one is much more sophisticated, designed to perform a wiser exploration of the search space: the MDL-based fitness function. Finally, in order to avoid situations such as the represented in figure 8.1, where there is an irreversible collapse of the population, a penalty function added to the fitness formula will be described.

8.5.1 Hierarchical selection operator

Motivation and definition

As described in the introduction of this chapter, this first generalization pressure method is very simple in concept. It consist of a comparison function integrated into a tournament selection that decides which individual is the winner of each tournament. Its inspiration is the *Occam's razor* principle, which informally says that given two equally accurate solutions, keep the simplest one. In the case of this operator, the *equally accurate* has been changed to *similarly accurate*. The code of the hierarchical selection is represented in figure 8.5.

Why is the rationale of this *similarly accurate*, which introduces an extra parameter? ($threshold$) if the threshold were 0, it would mean that the best individual of the population is the one achieving most training accuracy with the minimum number of rules. This is totally correct for synthetic or totally consistent domains, where it is normal to achieve perfect accuracy, but what happens if there is noise in the dataset? The hierarchical selection would be unable to stop the system learning specific rules that cover wrong training examples, because they would increase the training accuracy of the generated solution.

On the other hand, if we have a threshold slightly larger than 0, the system will be able to learn *almost* the maximum possible training accuracy, but it will avoid learning rules that have an almost insignificant contribution to the training accuracy, which usually means inconsistent instances or noise. There is an open question, which is how do we tune $threshold$. In previous work (Bacardit & Garrell, 2002c), the experimentation showed that value 0.01 was quite good in general for real datasets, which smaller values such as 0.001 were better for synthetic ones, which is consistent with the rationale of the operator stated above. As the experimentation reported in this chapter only contains real problems, for the sake of simplicity we will only use value 0.01 for the threshold.

Figure 8.5: Code of the hierarchical selection operator

Hierarchical selection operator
Input : $Indiv_a, Indiv_b, threshold$
$winner = NULL$
If $|Indiv_a.accuracy - Indiv_b.accuracy| < threshold$
  If $Indiv_a.numRules < Indiv_b.numRules$
    $winner = Indiv_a$
  Else If $Indiv_a.numRules > Indiv_b.numRules$
      $winner = Indiv_b$
  EndIf
EndIf
If $winner$ is $NULL$
  If $Indiv_a.fitness > Indiv_b.fitness$
    $winner = Indiv_a$
  Else If $Indiv_a.fitness < Indiv_b.fitness$
      $winner = Indiv_b$
  Else
    If random$[0, 1] < 0.5$
      $winner = Indiv_a$
    Else
      $winner = Indiv_b$
    EndIf
  EndIf
EndIf
Output : $winner$

In order to integrate the operator with the rest of the system, some modifications are needed:

- As the default tuning for the rule-deletion operator leaves a small subset of useless rules in the individual, to act as neutral code, it is logical to exclude these rules from the operator. Thus, the number of rules of an individual used in the operator will include only the alive rules.

- For the *ADI* knowledge representation we need an alternative definition of complexity to using simply the number of alive rules, because now there are to factors of complexity: number of rules and number of intervals per attribute. The simplest alternative is to use as the length of an individual the total sum of intervals contained in the rules of the individual, that serves for both factors.

Behavior of the hierarchical selection operator

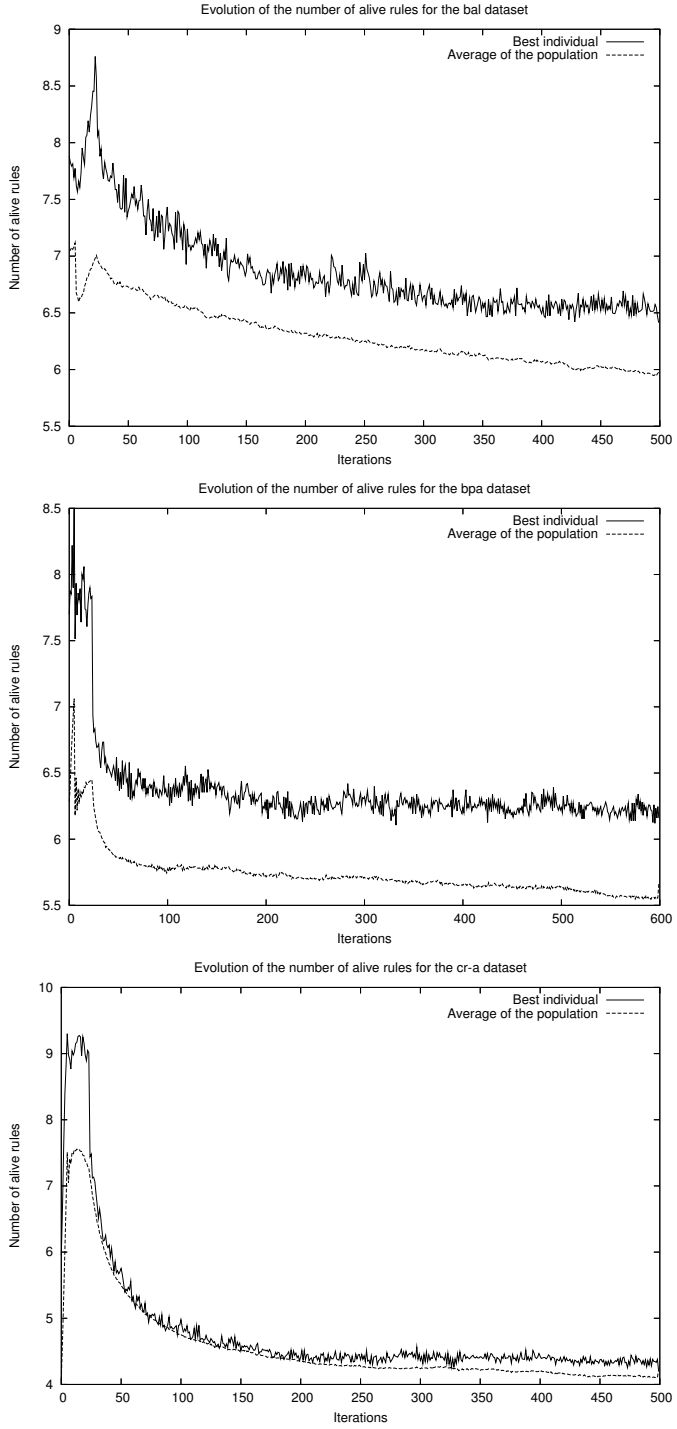
What is the behavior of the hierarchical selection operator? section 8.6 will show performance and run-time results of the operator over a large set of problems. Here the aim is to illustrate the general tendency (in the complexity of the individuals) that the operator shows through the iterations. Figure 8.6 shows the evolution, through the iterations, of the number of alive rules of the best individual and the average of the population for the *bal*, *bpa* and *cr-a* datasets.

The hierarchical selection method uses a specific-to-general policy. In the early iterations of the learning process it frequently finds new solutions that outreach the previous best accuracy by more than $threshold$. In this situation the number of rules of the individuals is irrelevant. But as the learning curve stabilizes, the differences in accuracy between the bests individuals of the population become smaller than $threshold$. Then, the smaller individual are mostly selected and, as a consequence, the size of the individuals slowly decreases.

8.5.2 The MDL-based fitness function

This subsection describes the alternative method to the hierarchical selection proposed in this thesis to apply generalization pressure in the learning system. It is inspired in the *Minimum Description Length* (*MDL*) principle (Rissanen, 1978) which is an interpretation of the Occam's Razor principle based on the idea of data compression, that takes into account both the simplicity and predictive accuracy of a theory. Pfahringer (Pfahringer, 1995) did a very good and brief introduction of the principle:

Figure 8.6: Evolution of number of alive rules for the *bal*, *bpa* and *cr-a* datasets with the hierarchical selection operator



> Concept membership of each training example is to be communicated from a sender to a receiver. Both know all examples and all attributes used to describe the examples. Now what is being transmitted is a theory (set of rules) describing the concept and, if necessary, explicitly all positive examples not covered by the theory (the false-negative examples) and all negative examples erroneously covered by the theory (the false-positive examples). Now the cost of a transmission is equivalent to the number of bits needed to encode a theory plus its exceptions in a sensible scheme. The MDL principle states that the best theory derivable from the training data will be the one requiring the minimum number of bits.

For the exact definition of the fitness function used, we have taken the general formula used in C4.5rules (Quinlan, 1993):

$$MDL = W \cdot \text{theory bits} + \text{exception bits} \tag{8.2}$$

The objective of the GA is to minimize this function. W is a weight that adjust the relation between theory and exception bits. The length of the theory bits (TL) is defined as follows:

$$TL = \sum_{i \in alive(individual)} TL_i \tag{8.3}$$

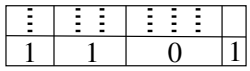
Where $alive_rules(individual)$ is the subset of rules of the individual that are alive. The definition of the rules for all the knowledge representations used share a common structure: $condition \rightarrow class$. The condition is defined as a certain structure (usually a predicate) formed by elements, each of them associated to an attribute of the problem. Therefore, TL_i is defined as follows:

$$TL_i = \sum_{j=1}^{\text{na}} TL_i^j. \tag{8.4}$$

Where *na* is the number of attributes of the problem. TL_i^j is the length of the predicate associated to the attribute j of the rule i, and has a specific formula for each knowledge representation used. The reader can see that we have omitted a term in the formula related to the class associated to the rule. As it is a value common for all the possible rules it becomes irrelevant and it has been removed for simplicity reasons.

The exceptions part of the *MDL* principle (EL) represents the act of sending the class for the misclassified or unclassified examples to the receiver. We implement this idea by sending the number of exceptions plus, for each exception, its index in the examples set (supposing

Figure 8.7: Example of an ADI2 attribute predicate



that sender and receiver have the examples organized in the same order) and its class:

$$EL = log_2(ne) + (nm + nu) \cdot (log_2(ne) + log_2(nc)) \tag{8.5}$$

Where ne is the total number of examples, nm is the number of wrongly classified examples, nu is the number of unclassified examples and nc is the number of classes of the problem.

Adaptation of the *MDL* principle for each knowledge representation

The length of the predicate associated to each attribute (TL_i^j) has to be adapted to the type of the attribute and the knowledge representation. While designing the formula to calculate this length we have to remember that the philosophy of the *MDL* principle is to promote simple but accurate solutions. Therefore, we will prefer formula definitions that promote bias towards simpler solutions although there may exist shorter/simpler definitions.

MDL formula for real-valued attributes and ADI knowledge representation The predicate associated to an attribute by this representation is defined as a disjunction of intervals, where each interval is a non-overlapping number of *micro-intervals* and can take a value of either true of false. Thus, the information to transmit is the number of intervals of the predicate plus, for each interval, its size and value (1 or 0):

$$TL_i^j = log_2(\mathrm{MaxI}) + ni_i^j \cdot (log_2(\mathrm{MaxMI}) + 1) \tag{8.6}$$

$MaxI$ is the maximum number of intervals allowed in a predicate, ni is the actual number of intervals of the predicate and $MaxMI$ is the maximum allowed number of *micro-intervals* in the predicate.

Given the example of attribute predicate in figure 8.7, where we have 4 intervals , and supposing that the maximum numbers of intervals and *micro-intervals* are respectively 5 and 25, its MDL size is defined as follows:

$$TL_i^j = log_2(5) + 4 \cdot (log_2(25) + 1)$$

MDL formula for real-valued attributes and unordered bounds intervals representation (UBR) The predicate associated to an attribute by this representation is a real-valued interval encoded as the two bounds of the interval (without fixed ordering) and two bits that represent the relevance state of each of the two bounds of the interval. The aim of the MDL formulation for this representation is to promote irrelevant intervals. Thus, we send the relevance state of each interval bound (one bit per bound) and, if necessary, the value of the bound. What is the description length of a bound? We use the simple solution of using the length of a *float* data type.

$$\begin{aligned} TL_i^j &= 2 \\ &+ \begin{cases} size(float) & \text{if } lowerBound_i^j \text{ is relevant} \\ 0 & otherwise \end{cases} \\ &+ \begin{cases} size(float) & \text{if } upperBound_i^j \text{is relevant} \\ 0 & otherwise \end{cases} \end{aligned} \tag{8.7}$$

Given an attribute in the $[0..1]$ domain, and the three following predicates related to this attribute:

$$\begin{aligned} p1 &: false, false, [0, 2..0, 6] \\ p2 &: false, true, [0.6..0.3] \\ p3 &: true, true, [0.1..0.9] \end{aligned}$$

The relevance bits make the whole interval $p3$ and the upper bound of $p2$ be totally irrelevant. Thus, the theory length of these intervals are:

$$\begin{aligned} p1 &: TL_i^j = 2 + 2 \cdot size(float) \\ p2 &: TL_i^j = 2 + size(float) \\ p3 &: TL_i^j = 2 \end{aligned}$$

MDL formula for discrete attributes and GABIL representation The predicate associated to an attribute by this representation is defined as a disjunction of all the possible values that can take the attribute. The simpler way of transmitting this predicate is sending the binary string that the representation uses to encode it. This is the approach used by Quinlan in C4.5rules (Quinlan, 1993). However, this definition does not take into account the complexity of the term and does not provide a bias towards generalized solutions.

Therefore, we define a different formula which is very similar to the one proposed for the *ADI2* knowledge representation. In this formula we simulate that we have merged the neighbor values of the predicate which have the same value (true or false):

$$TL_i^j = log_2(nv_j) + 1 + ni_i^j \cdot log_2(nv_j) \tag{8.8}$$

nv is the number of possible values of the attribute j and ni is the number of "simulated intervals" that exist in the predicate. The only difference between this formula and the *ADI2* one is that we do not have to transmit the value of all the "simulated intervals", but only the first one (one bit).

If we had an attribute predicate such as "1111100001" we can see that we have 10 values and 3 "simulated intervals" and that the MDL size of the predicate would be:

$$TL_i^j = log_2(10) + 1 + 3 \cdot log_2(10)$$

This approach makes sense for ordinal attributes, where an order between values exists, but not for nominal ones. However, we think that this definition is also useful for nominal attributes because we want to promote generalized predicates, where most of the values are true, and this means having few "simulated intervals".

MDL formula for discrete attributes and XCS representation The predicate for this representation is a conjunction of tests where each test is either a possible value of the attribute or "don't care" (#). In order to promote generalized rules, the message for a # symbol should be much shorter that the message for the other symbols. Our proposal is to send a bit which determines the relevance of the predicate and, if necessary, the value of the predicate test.

$$TL_i^j = 1 + \begin{cases} log_2(numValuesAttr_j) & \text{if } value_i^j \text{is relevant} \\ 0 & otherwise \end{cases} \tag{8.9}$$

Given a whole rule defined as "#12#1|1" with 5 attributes. Attribute 1,2 and 3 can take 3 different values. Attributes 4 and 5 can take 7 different values. The MDL size for the rule for this rule would be:

$$TL_i = \sum_{j=1}^{5} TL_i^j = 1 + (1 + log_2(3)) + (1 + log_2(3)) + 1 + (1 + log_2(7))$$

Parameter-less *MDL* principle

If we examine all the formulas of the *MDL* principle we only find one parameter: W, which adjusts the relation between the length of the theory and the length of the exceptions. Quinlan used a value of 0.5 for this parameter in *C4.5rules* and reported the following in page 53 of (Quinlan, 1993):

> Fortunately, the algorithm does not seem to be particularly sensitive to the value of W.

Unfortunately, our environment of application of the *MDL* principle is quite different and the value of the W parameter becomes very sensitive. The reason of this fact is the selection pressure of the *GA*. If this weight is to high, the individuals may collapse into one-rule solutions, as represented in figure 8.1.

This problem with the adjusting of W leads to a question: Is it possible to find a good method to adjust this parameter automatically? The completely rigorous answer, being aware of the *No Free Lunch Theorem* (Wolpert & Macready, 1995) and the *Selective Superiority Problem* (Brodley, 1993) is no.

Nevertheless, at least we can try to find a way to automatically make the system perform "quite well" in a broad range of problems. In order to achieve this objective we have developed a simple approximation which starts the learning process with a very strict weight (but loose enough to avoid a collapse of the population) and relaxes it through the iterations when the *GA* has not found a better solution for a certain number of iterations. This method can be represented by the code in figure 8.8.

$InitialRateOfComplexity$ defines which percentage of the MDL formula should the term $W \cdot TL$ have. Using this supposition and given one individual from the initial population, we can calculate the value of W. We have used a simple policy to select this individual: the one with more training accuracy ($W = \frac{InitialRateOfComplexity \cdot EL}{(1-InitialRateOfComplexity) \cdot TL'}$).

This raises a question, is this individual good enough? If we recall section 8, it is more probable that this individual will be long than short. Then, maybe we would be initializing W with too mild a value. Therefore, before calculating the initial value of W we do a last step: scaling the theory length of this individual ($TL' = TL \cdot \frac{NR}{NC}$), using as a reference the minimum possible number of rules of an optimal solution: the number of classes of the domain.

We can see that in order to automatically adjust one parameter we have introduced three extra parameters (*InitialRateOfComplexity*, *MaximumBestDelay* and *WeightRelaxationFactor*). The second parameter is easy to setup if we consider the takeover time for the tournament selection (Goldberg & Deb, 1991). Given a tournament size of 3 and a population size of 300,

Figure 8.8: Code of the parameter-less learning process with automatically adjusting of W

Initialize GA
$Ind =$ Individual with best accuracy from the initial GA population
$TL =$ Theory Length of Ind
$EL =$ Exceptions Length of Ind
$NR =$ Number of rules of Ind
$NC =$ Number of Classes of the domain
$TL' = TL \cdot \frac{NR}{NC}$
$W = \frac{InitialRateOfComplexity \cdot EL}{(1-InitialRateOfComplexity) \cdot TL'}$
$Iteration = 0$
$IterationsSinceBest = 0$
While $Iteration < NumIterations$
    Run one iteration of the GA using W in fitness computation
    If a newbest individual has been found **then**
        $IterationsSinceBest = 0$
    Else
        $IterationsSinceBest = IterationsSinceBest + 1$
    EndIf
    If $IterationsSinceBest > MaximumBestDelay$ **then**
        $W = W \cdot WeightRelaxationFactor$
        $IterationsSinceBest = 0$
    EndIf
    $Iteration = Iteration + 1$
EndWhile

Table 8.1: Tests with the MX-11 domain done to find the values of InitialRateOfComplexity (*IROC*) and WeightRelaxationFactor (*WRF*)

WRF	IROC	Test acc.	Num. of Rules	Iterations until perfect accuracy
0.7	0.05	100.0±0.0	9.3±0.6	301.4±56.8
	0.075	100.0±0.0	9.2±0.5	309.0±62.6
	0.1	100.0±0.0	9.2±0.5	333.3±62.2
0.8	0.05	100.0±0.0	9.3±0.5	331.0±71.5
	0.075	100.0±0.0	9.2±0.3	364.4±75.3
	0.1	100.0±0.0	9.2±0.5	374.3±66.9
0.9	0.05	100.0±0.0	9.2±0.5	428.6±99.7
	0.075	100.0±0.0	9.2±0.4	475.5±95.6
	0.1	100.0±0.0	9.1±0.4	518.4±110.2

the takeover time is 6.77 iterations. Considering that we have both crossover and mutation in our *GA*, setting $MaximumBestDelay$ to 10 seems quite safe.

Setting $InitialRateOfComplexity$ is also relatively easy: if the value is too high (giving too much importance to the complexity factor of the MDL formula) the population will collapse. Therefore, we have to find the maximum value of $InitialRateOfComplexity$ that lets the system perform a correct learning process. Doing some short tests with various domains we have seen that values over 0.1 are too much dangerous. In order to adjust this parameter more finely and also set *WeightRelaxationFactor* we have done tests using again the *MX-11* domain testing three values of each parameter: 0.1, 0.075 and 0.05 for $InitialRateOfComplexity$ and 0.9, 0.8 and 0.7 for $WeightRelaxationFactor$.

The results can be seen in table 8.1, showing three things: test accuracy and the number of rules of the best individual in the final population and also the average iteration where 100% training accuracy was reached. We can see that all the tested configuration manage to reach a perfect accuracy, and also that the number of rules of the solutions are very close to the optimum 9 ordered rules. The only significant differences between the combinations of parameters tested comes when we observe the iterations needed to reach 100% training accuracy. We can see that as more mild are the parameters used, fewer iterations are needed. This brings up the question of how well can this behavior be extrapolated to other domains. We have to be aware that *MX-11* is a synthetic problem without noise.

In order to check how the system is behaving in real problems, we repeated this test with the *wbcd* dataset. The results can be seen in table 8.2. Iterations are not included in this table because we do not know the ideal solution for this problem. Instead, we have included training accuracy. It will help illustrate the completely different landscape that we have here: Although the differences are not significant, we can see that the more mild the parameters used, we have

Table 8.2: Tests with the *bre* domain done to find the values of InitialRateOfComplexity (*IROC*) and WeightRelaxationFactor (*WRF*)

WRF	IROC	Training acc.	Test acc.	Num. of Rules
0.7	0.05	98.2±0.3	95.6±1.5	4.3±1.5
	0.075	98.2±0.3	95.8±1.5	4.1±1.3
	0.1	98.1±0.3	95.9±1.7	3.9±1.2
0.8	0.05	98.1±0.3	95.8±1.5	3.9±1.3
	0.075	98.0±0.3	96.0±1.7	3.7±0.8
	0.1	97.9±0.3	96.0±1.7	3.5±0.9
0.9	0.05	97.8±0.3	95.9±1.7	2.9±0.9
	0.075	97.6±0.3	96.0±1.8	2.3±0.6
	0.1	97.5±0.3	95.9±1.8	2.2±0.5

more training accuracy, more rules, and less test accuracy. It seems quite clear that the system suffers from over-learning if its working parameters are not strict enough. The results showed here are illustrative of the general behavior of *MDL* in several datasets. Therefore, we select 0.075 and 0.9 as the values of $InitialRateOfComplexity$ and $WeightRelaxationFactor$ respectively for the rest of experimentation in this chapter.

Before showing the results for all the datasets tested it would be interesting to see the stability of the W tuning heuristic presented in this section. In figure 8.9 we can see the evolution of W through the learning process for the *wbcd* and *prt* problems The values in the figure have been scaled in relation to the initial W value. These two problems are selected because they show two alternative behaviours due to having quite different number of rules in their optimal solutions. We can see that the differences in the evolution of W for different executions shrink through the iterations, showing the stability of the heuristic.

Behavior of the MDL-based fitness function

As made for the hierarchical selection, we want to illustrate the general tendency (in the complexity of the individuals) that the operator shows through the iterations. Figure 8.10 shows the evolution, through the iterations, of the number of alive rules of the best individual and the average of the population for the *bal*, *bpa* and *cr-a* datasets.

The behavior of *MDL* is quite different from hierarchical selection , because of the behaviour of the W control heuristic. This method starts the learning process giving much importance to the size of the individual, and relaxes this importance through the iterations as dictated by the heuristic. Therefore, the behaviour is general to specific.

In figure 8.10 we can also see the main problem of the *MDL* method, which is the over-relaxation of the W weight. The philosophy of the algorithm we have proposed to tune W

Figure 8.9: Evolution of W through the learning process

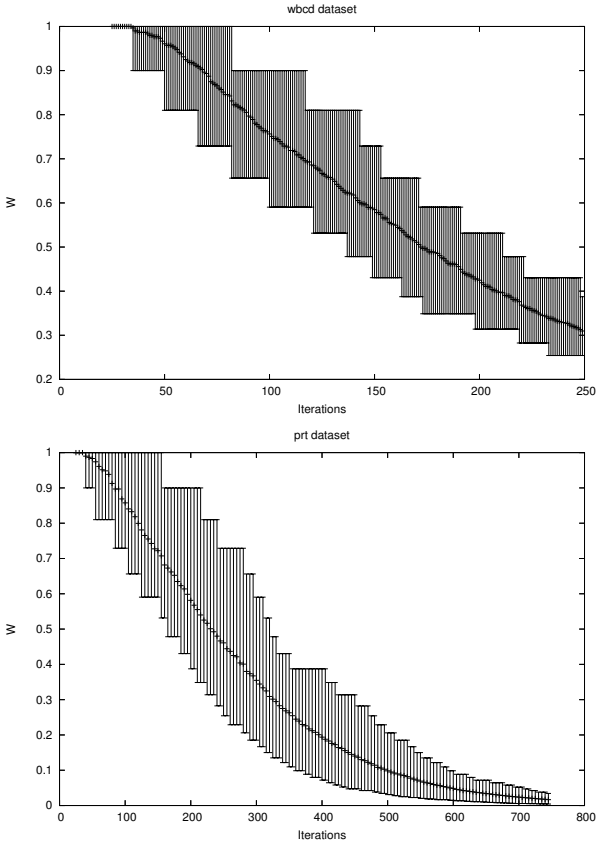
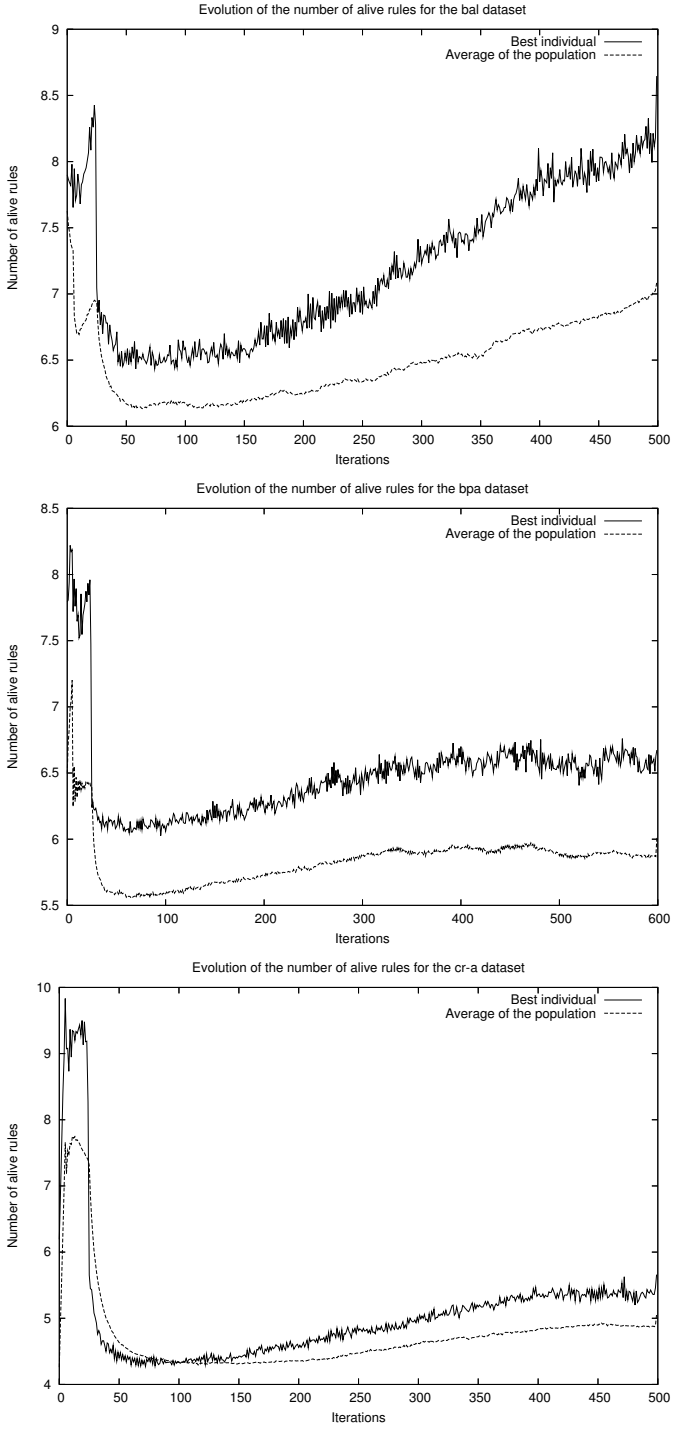


Figure 8.10: Evolution of number of alive rules for the *bal*, *bpa* and *cr-a* datasets with the hierarchical selection operator



is that we relax this weight when it is too strict, that is, when the *GA* cannot find a better individual for a certain number of iterations. This condition is sometimes difficult to control, and maybe if the system was given more iterations, the test performance in some domains would decrease. This issue needs more work.

8.5.3 Tuning the generalization pressure methods for proper learning

It can be observed in figure 8.10 for the MDL-based fitness function, as well as in figure 8.6 for the Hierarchical selection that the behavior of the system changes drastically at iteration 25. The reason is simple: like the rule-deletion operator, it is not wise to start using these techniques from the initial iteration, because we can produce a population collapse like the represented in figure 8.1. Thus, the two operators are activated at iteration 25.

Another question arises: Are 25 iterations enough to guarantee that the population does not collapse? The answer is yes for most datasets. However it is not a guarantee that the learning process is performed properly, because we do not have any guarantee that good rules that have been forgotten because of the activation of the generalization pressure operator are learned again. As an example of this problem, figure 8.11 shows, for the *mmg* dataset, a correlation between the average individual size of each run in the test and the achieved test accuracy. The figure shows clearly how the runs that evolve smaller individuals because of the generalization pressure end up having less test accuracy.

Thus, some mechanism that prevents the system from reducing too much the number of active rules in the population is needed. This mechanism takes the form of a penalty formula applied to the fitness function. In this way, we can force the system to discard individuals that can lead to an incorrect learning process. This penalty function is applied if the number of alive rules in the individual falls below a certain threshold. The penalty function is applied with the code in figure 8.12. The penalty applied is relative to how far the number of alive rules is from the threshold. Also, depending on the fitness function used (squared accuracy for hierarchical selection or the MDL one), the penalty is multiplied (fitness is maximized) or divided (fitness is minimized) by the raw fitness.

Now the question is the tuning of $minThreshold$. It is very difficult to decide a global policy applied on-line, thus, for the experimentation reported in this chapter, the value of $minThreshold$ was adjusted manually for each dataset. The procedure followed is to initialize the threshold to 2, and perform some short runs to detect is its learning process is being constrained by the size of the individuals. If this situation is detected, the threshold is increased by 1, and the procedure is repeated. The adjusting procedure stops when the parameter reaches value 6, because higher values could start to constrain too much the learning process. This

Figure 8.11: Correlation between the average number of alive rules per individual and the test accuracy for the *mmg* dataset, if no penalty function is used

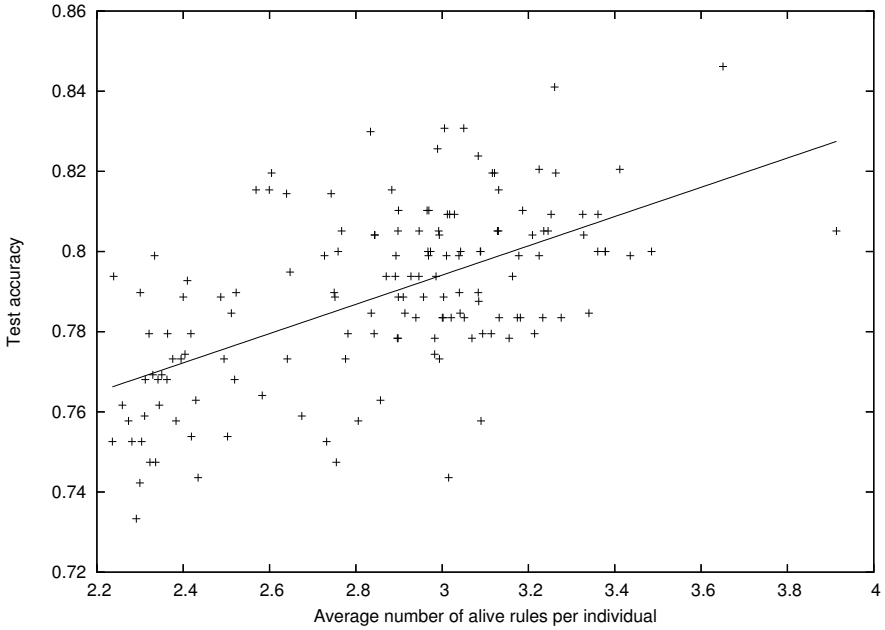


Figure 8.12: Code of the penalty function used to avoid a population collapse

Small individuals penalty function
Input : $Individual, minThreshold$
$aliveRules = Individual.aliveRules$
$penalty = 1$
If $aliveRules < minThreshold$
  $penalty = (1 - 0.05 * (minThreshold - aliveRules))^2$
EndIf
$rawFitness = Individual.rawFitness$
If hierarchical selection is used
  $finalFitness = rawFitness * penalty$
Else If MDL-based fitness function has been used
  $finalFitness = rawFitness/penalty$
EndIf
$Individual.fitness = finalFitness$
Output : $Individual$

aim of this penalty function is to apply a small fitness decrease to individuals with low number of rules, not to influence dramatically the fitness function.

8.6 Comparing experimentally the generalization pressure methods

This section describes the extensive experimentation done to analyze the behavior of the two alternative generalization pressure methods described in previous section: the hierarchical selection and the MDL-based fitness function. These two techniques will be used in combination with the rule pruning operator, and the fitness penalty formula also described in previous section.

The aim of these experiments is not only to determine which of these two methods is the best, but also to study the behavior showed by each method over a large set of problems. For this reason, the tests include many different scenarios. First , the four knowledge representations for which an MDL-formula has been proposed (*ADI* and *UBR* for real-valued attributes and the *GABIL* and *XCS* representations for nominal attributes) are tested. Next, it would be interesting to know if these techniques can be combined with methods that have also showed to introduce extra generalization pressure, like the *ILAS* windowing scheme. Thus for each knowledge representation two configurations of *ILAS* will be used, using one and two strata with constant iterations strategy. No comparative tests will be done between settings of *ILAS* or between the knowledge representations. This has already been done in previous chapters of this thesis. Here we are only interested in extracting some patterns of behaviour of each generalization pressure methods, and determine if these patterns change in different scenarios.

Also, to have an external reference of another recent generalization pressure method used in a Pittsburgh GBML system, we include in the experimentation the *MOLCS* method (Llorà, Goldberg, Traus, & Bernadó, 2002) based on multi-objective optimization (MOO) described in section 3.6. What has been used in the experimentation is a reimplementation of the technique inside the framework of *GAssist*. The introduction of this generalization pressure method in the experimentation forces us to disable the default rule mechanism defined in chapter 5 that has been used in the experimentation of all the other chapters. The reason is that the automatic determination of default class method cannot work together with the MOO because both techniques use modified selection algorithms, that cannot be mixed easily. Also, the fitness penalty formula will not be applied here, because it totally contradicts the philosophy of MOO, whose aim is to evolve solutions that cover all the Pareto front. We would like to remark, before showing the results of this experimentation, that the results of the *MOLCS* system

Table 8.3: Settings of GAssist for the experimentation with generalization pressure methods

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	300
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 1000
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	disabled
GABIL knowledge representation	
Probability of ONE	0.75
XCS knowledge representation	
Probability of ♯	0.75
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Probability of Reinitialize (begin,end)	(0.02,0)
Maximum number of intervals	5
Uniform-width discretizers used	4,5,6,7,8,10,15,20,25 bins
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90
Hierarchical selection operator	
Iteration of activation	25
Threshold	0.01

should be treated with a grain of salt. The environment where this system was designed is different from *GAssist* in different aspects, especially the knowledge representations. Thus, the conclusions that might be extracted from the experimentation in this chapter should be treated as how this MOO technique works in the framework of GAssist, not as how good it is in general.

All tests will use the configuration described in table 8.3.

As there are several datasets that contain a mix of real-valued and nominal attributes, we have decided to use together the *ADI* and *GABIL* representation on one hand and *UBR* and

Table 8.4: Results averages of the generalization pressure methods experimentation for the ADI and GABIL representations without ILAS

Method	Training acc.	Test acc.	#rules
MDL	89.56±11.82	81.41±14.04	6.68±3.01
Hierar	88.83±12.45	80.94±14.00	5.79±2.44
MOLCS	91.23±10.66	80.59±14.22	9.67±4.30

XCS on the other hand. The rationale of these two alternative groups is to compare rules with two different kind of predicates. Conjunctive normal form predicates for the first group of representations and totally conjunctive predicates on the other hand. Also, the inspiration of the MDL formula for each group of representations is very similar.

Finally, the following subsection will show the average results over all datasets of the tested configurations and the statistical tests applied to the results. The results for each dataset are placed in appendix C.

8.6.1 Experimentation with ADI and GABIL representations and 1 stratum

Table 8.4 contains the average results of the experimentation of this subsection. These averages show how the three methods have three different degrees of generalization pressure: *Hierar* is the method applying more pressure, reflected in the lowest training accuracy and number of rules (at least with the current setting of 0.01 for its threshold). Then, MDL shows a middle-point pressure degree (more training accuracy and more number of rules). Finally, the results show how MOLCS is the system applying less pressure, reflected by the top training accuracy and number of rules.

However, looking at the test accuracy it can be observed that the top training accuracy of MOLCS does not translate into test accuracy, reflecting that the system is suffering from over-learning. On the other hand, the MDL method is the one achieving most test accuracy, indicating that it is applying the correct degree of generalization pressure. Considering the performance of the hierarchical selection operator, we can affirm that the system is not learning properly because there is too much generalization pressure and, consequently, the solutions generated are over-general.

Statistical t-tests were applied to these results, and are summarized in table 8.5. The t-tests show how the ranking of test accuracy is equivalent also to robustness, being *MDL* the most robust method, and the *MOLCS* the most weak.

Table 8.5: Results of the t-tests applied to the results of the experimentation on generalization pressure methods using ADI and GABIL representations without ILAS, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	MDL	Hierar	MOLCS	Total
MDL	-	4	8	12
Hierar	0	-	4	4
MOLCS	2	2	-	2
Total	2	6	12	

Table 8.6: Results averages of the generalization pressure methods experimentation for the ADI and GABIL representations using ILAS with 2 strata

Method	Training acc.	Test acc.	#rules
MDL	88.20±12.19	81.41±13.93	5.64±2.13
Hierar	88.09±12.52	81.12±14.16	5.27±2.02
MOLCS	87.41±12.14	79.76±14.63	7.24±2.90

8.6.2 Experimentation with ADI and GABIL representations and 2 strata

Table 8.6 contains the average results of the experimentation of this subsection. The situation here is quite different from the results in the previous subsection. The use of the ILAS windowing system benefits more the *Hierar* method than MDL, and now the accuracy different between both methods is smaller. Also, the results show how MOLCS does not combine well with ILAS. Now MOLCS has the lowest average training accuracy, reflecting that it is not learning properly.

Probably the reason for this is that it cannot benefit anymore from the elitism mechanism used in MOLCS, that copies the best 30% (in accuracy) from the prior population to the current one. This elitism gap is ineffective now because these individuals were selected based on a fitness function that is not used in the current iteration. Therefore, the elitism mechanism which was beneficial without ILAS, is now only a source of noise.

Statistical t-tests were applied to these results, and are summarized in table 8.7. The t-tests reflect even more the negative interaction between *MOLCS* and *ILAS*, being outperformed significantly by *MDL* in almost half of the datasets used in the experimentation, and by *Hierar* in 10 out of 25 datasets. The tests also show how *MDL* and *Hierar* perform very similarly, with *MDL* being slightly superior.

Table 8.7: Results of the t-tests applied to the results of the experimentation on generalization pressure methods using ADI and GABIL representations with ILAS (2 strata), using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	MDL	Hierar	MOLCS	Total
MDL	-	1	12	13
Hierar	0	-	10	10
MOLCS	0	1	-	1
Total	0	2	22	

8.6.3 Experimentation with UBR and XCS representations and 1 stratum

Table 8.8 contains the average results of the experimentation in this subsection. These averages show how, comparing them to the experimentation with the other two knowledge representations, *MOLCS* increases its performance while the other two methods have its performance degraded. *MOLCS* still shows some over-learning, if it is compared to *MDL*, having lower training accuracy but better test accuracy. The *Hierar* method it the one having more accuracy drop, almost 1% which, considering that it is an average over all datasets, is important.

Clearly, the hierarchical selection operator is less effective in promoting accurate but compact individuals in this representation. Comparing the average training accuracy of *Hierar* for both kinds of knowledge representations (88.83$\pm$12.45 for *ADI* and *GABIL* versus 89.19$\pm$12.17 for *UBR* and *XCS*) illustrate this problem. The *UBR* knowledge representation has more exploration power that *ADI*, which is constrained by the information loss introduced by the discretization algorithms. Therefore, *UBR* can achieve better training accuracy. However, this might mean that *UBR* has more probabilities of learning the noise contained in real datasets, leading to a lower test accuracy. It is the job of the generalization pressure method to avoid this situation, but the *hierarchical selection operator* seems to be unable to perform properly this task.

The hierarchical selection can only be effective if the difference between the accuracies of the well generalized solutions and the ones containing over-learning is small. If this gap is enlarged by using a better exploration mechanism, it is not effective anymore.

The statistical tests applied to these results, summarized in table 8.9, show how now *Hierar* and *MOLCS* have similar levels of performance and robustness. The tests also show how *MDL*

Table 8.8: Results averages of the generalization pressure methods experimentation for the UBR and XCS representations without using ILAS

Method	Training acc.	Test acc.	#rules
MDL	89.42±11.68	81.22±13.93	8.48±3.62
Hierar	89.19±12.17	79.99±13.93	8.04±4.01
MOLCS	89.76±12.44	80.25±14.45	10.36±4.46

Table 8.9: Results of the t-tests applied to the results of the experimentation on generalization pressure methods using UBR and XCS representations without ILAS, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	MDL	Hierar	MOLCS	Total
MDL	-	6	6	12
Hierar	1	-	3	4
MOLCS	0	3	-	3
Total	1	9	9	

is better than both.

8.6.4 Experimentation with UBR and XCS representations and 2 strata

Table 8.8 contains the average results of the experimentation in this subsection. These results maintain the same tendencies showed by the other two representations when *ILAS* was activated: *MDL* show similar performance, *Hierar* improves its performance and *MOLCS* degrades it. This set of tests does not introduce any new interesting observation, but it is consistent with the behavior identified by the previous tests.

The statistical tests applied to these results, summarized in table 8.9, show how now there is again a clear ranking between the methods. *MDL* is significantly better than *Hierar* and *Hierar* is significantly better than *MOLCS*. This behavior is different from the observed with the other two knowledge representations when *ILAS* was used, where *MDL* and *Hierar* showed very similar performance.

Table 8.10: Results averages of the generalization pressure methods experimentation for the UBR and XCS representations using ILAS with 2 strata

Method	Training acc.	Test acc.	#rules
MDL	87.72±12.09	81.16±14.04	6.58±2.58
Hierar	88.27±12.25	80.24±13.79	6.65±2.19
MOLCS	86.63±13.13	79.64±14.59	7.20±2.61

Table 8.11: Results of the t-tests applied to the results of the experimentation on generalization pressure methods using UBR and XCS representations with ILAS (2 strata), using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	MDL	Hierar	MOLCS	Total
MDL	-	4	9	13
Hierar	0	-	3	3
MOLCS	0	1	-	1
Total	0	5	12	

8.7 Discussion and further work

The results in the previous section seem to show a very clear conclusion: *MDL* is superior or equal to the other two tested generalization pressure methods in all the scenarios included in the experiments. The reality, however is different: so far the only clear thing is that *MDL* is better than the other methods in the exact settings used in this experimentation. Can it be affirmed with high confidence that these results can be generalized to other settings and scenarios?

How can the performance of the *hierarchical selection operator* be improved? Looking at the experimentation with *ADI* and *GABIL* representations, the experimentation showed that it was not learning properly because it was applying too much generalization pressure. To fix this problem two alternatives could be considered:

- Increase the number of iterations to achieve a proper learning level
- Change the value of $threshold$ from 0.01 to something lower.

The first alternative has two problems: the first one is obviously the extra computational cost of the method. The second one is that there is no guarantee that extra iterations will lead

to a proper learning, if the pressure applied blocks any further accuracy improvement of the population. The second alternative clearly would allow the system to achieve better training accuracy. However it is necessary to question if this more relaxed generalization pressure benefits equally all datasets. What happens if in some datasets a lower threshold makes the method ineffective to avoid over-learning?

On the other hand, if we look at the results of the *UBR* and *XCS* representations, the *hierarchical selection operator* shows the totally opposite behavior: it suffers from over-learning. How could we fix this problem? With the opposite solutions to the ones considered so far: lower the number of iterations or increase $threshold$. Therefore, it can be affirmed with confidence that the *hierarchical selection operator*, unlike *MDL*, cannot perform well on all the tested knowledge representations with a single set of parameters.

Nevertheless, using a single set of parameters for each knowledge representation is quite an acceptable solution. Thus, more tests are required to determine if it is possible to find another set of parameters that makes the *hierarchical selection operator* perform well on most datasets for each knowledge representation.

The results of *MOLCS* showed two things: First, it seems that it cannot be combined with *ILAS*, because of the elitism mechanism that it uses. This is a major drawback, because it means that in order to perform properly, *MOLCS* needs almost the double of run-time than *MDL*. The previous chapter showed how *ILAS* is beneficial for the performance of the system in many different scenarios. A generalization pressure method that cannot be combined with it automatically is less interesting to use.

Independently of *ILAS*, can the performance of *MOLCS* be improved without major changes to the method? The tests showed how it suffers from over-learning, compared to *MDL*, in both kinds of knowledge representations. We have two hypothesis of how can this problem be fixed:

- Reducing the percentage of population included in the elitism gap of *MOLCS* from 30% to something lower. As this elitism set contains the individuals with best accuracy of the population, independently of their complexity, it is quite probable that some of these individuals suffer from over-learning.

- Obviously, reducing the number of iterations

These two alternatives have the same problems discussed before for the *hierarchical selection operator*. It needs to be determined with more tests if there is a unique set of parameters that works well on most datasets. It is important to remember that the number of iterations used in this experimentation for each dataset was determined using the procedure proposed for the *constant learning steps* strategy of *ILAS*, described in the previous chapter. Thus,

the *MDL* has a systematic method to tune the number of iterations used in each dataset. Can an alternative systematic procedure be developed for the *hierarchical selection operator* or *MOLCS*? This is another issue worth studying in further work.

The previous paragraphs have discussed about how the performance of the *hierarchical selection* or *MOLCS* can be improved, to achieve the performance level showed by *MDL*. However, another question arises. Can the performance of *MDL* also be improved? There are two elements of *MDL* that we think that can be:

- The heuristic procedure of automatic adjustment of W. This procedure is the reason that *MDL* performs well, compared to the other methods, with a single set of parameters. Thus, it is a very important and positive element of this method. However, it is quite complex, compared to the alternative generalization pressure methods. Can we find a way to determine the most suitable value of W for each dataset with a more simple procedure? Also, as it has been discussed before, the method lacks a criterion to determine when it is not necessary to modify the value of W anymore during the learning process. This is a potential source of over-learning in some datasets, if the weight ends up being too small to block learning wrong knowledge. The experiments showed that this method performs well compared to the alternatives, but we do not know if it has achieved its maximum performance. Maybe in some datasets the system suffers from over-learning, and maybe a good stop criterion for the adaptive procedure to adjust W could fix this problem.

- The formulation used to define the *theory length (TL)* for each knowledge representation. So far with the current TL definitions we have avoided answering a question: In the *ADI* representations, there are two different elements that need to be minimized: the number of rules in a rule set and the number of intervals per attribute. Does the current formulation give a proper balance between these two factors? If we give too much importance to the minimization of the intervals per attribute ration, the system can end up with too many irrelevant attributes (having only one interval). If we give too much importance to the minimization of the number of rules, the danger of a population collapse is higher. A proper balance is needed. It is necessary to determine if the current formulation (or an alternative one) provides this balance. The other knowledge representations have equivalent issues that need to be analyzed.

Finally, there is an important conclusion on the use of *MDL*. Analyzing the contents of the rules to extract a measure of complexity, instead of simply counting items in the individual (be this items rules, intervals, ...) has been shown to give better results so far, and this is the main difference between the generalization pressure method based on *MDL* proposed in this thesis to most of the methods reported in the literature. The *MDL* method probably is one

of the most complex methods existing, but it performs better, not only because it applies the correct degree of generalization pressure, but also because it takes into account the content of the individuals to compute the complexity measures, potentially providing a better method to explore the search space.

8.8 Summary of the chapter

This chapter focused on solving two problems very closely related. The first problem is a common issue in evolutionary computation techniques that use variable-length representations: the bloat effect. It consist in a growth without control of the size of the individuals. The way to control this problem in the framework of *GAssist* is a rule deletion operator that eliminates rules that do not contribute to the fitness of the individual, that is, that are useless. This operator, properly controlled can be beneficial in two aspects: run-time reduction and introduction of diversity.

The second problem is related to the machine learning field: the capacity of the learning system to generate well generalized solutions. Usually a well generalized solution is identified as an accurate solution of low complexity. Thus, the explicit control of the generalization issue is also closely tied to the control of the individuals size. Two alternative methods have been proposed in this thesis to apply generalization pressure in the system. The first one, the hierarchical selection operator, is very simple. The second one, the MDL-based fitness function, is much more complex.

These two methods, together with a recent alternative method reported in the literature were tested in a wide experimentation framework containing many different scenarios and datasets. The experiments showed how the MDL method was the best in all scenarios. Before concluding in general that MDL is the best method for a Pittsburgh GBML, it is necessary to guarantee that the alternative methods perform well. Several fixes have been proposed to achieve this objective.

Nevertheless, even if other methods can achieve similar performance than the MDL-based fitness function, this method is still relevant because of its novelty: it is one of the few methods in this area that considers the content of the individuals in guiding the exploration process, unlike most methods which only take into account performance (training accuracy) and very simple measures of complexity.

Chapter 9

The GAssist system: a global view and comparison

The four previous chapters have presented the contributions made by this thesis to the Pittsburgh model of genetic based machine learning. All techniques presented were tested using a wide range of datasets, followed by statistical tests. All this experimentation empirically determined which techniques were the best inside the framework of the GAssist *system.*

The aim of this chapter is to compare the performance of GAssist *(containing the best set of techniques/parameters) to several machine learning systems. This comparison will include several paradigms of machine learning, using different kinds of knowledge representation.*

With this this global comparison we do not want only to determine if GAssist *has competent performance compared to other modern learning systems, but also to determine why it has this performance. What is the cause of the low performance, in the datasets were it may be significantly outperformed and also what is the cause of the good performance, in the datasets where* GAssist *is significantly better.*

The chapter is structured as follows: Section 9.1 will provide a brief description of the machine learning systems included in the comparison. Next, section 9.2 will detail the configuration of GAssist *included in the comparison. Section 9.3 will summarize the experimentation done showing its results and the statistical tests applied to these results, followed by section 9.4 containing the analysis made of the results with the objective of explaining the good/poor performance of* GAssist*, compared to the other methods. Section 9.5 will contain the conclusions and further work of this experimentation and, finally, section 9.6 will provide a summary of the chapter.*

9.1 Description of the machine learning systems included in the comparison

The following systems are included in the comparison:

C4.5 (Quinlan, 1993) Is the well-known decision trees induction algorithm, descendant of *ID3* (Quinlan, 1986). As its predecessor, it uses an entropy-based criterion to decide which attribute and cut-point appears in the internal nodes, but includes also pruning techniques to discard over-specific parts of the tree.

IB1 (Aha, Kibler, & Albert, 1991) It is the simplest nearest-neighbour classifier technique. It uses the whole training set as the core of the classifier and euclidean distance to select the nearest instance to the new example, using its class as the prediction for the input instance.

IBk (Aha, Kibler, & Albert, 1991) It is an extension of the previous method. In this case, the k nearest instances to the input example are selected, and the class prediction provided by the system is the majority class in these k examples.

NaiveBayes (John & Langley, 1995) It is a very simple bayesian network approach that assumes that the predictive attributes are conditionally independent given the class and also that no hidden or latent attributes influence the prediction process. These assumptions allow for a very simple learning and predicting process. This version handles real-valued attributes by either using a gaussian estimator or a non-parametric kernel density estimator.

PART (Frank & Witten, 1998) This method generates rules from partially built decision trees, using the C4.5 methodology to build the trees. It also uses an on-line pruning process that works while the tree is being constructed, instead of applying it after the construction of the tree.

LIBSVM (Chang & Lin, 2001) This is a library containing implementations of support vector machine (SVM) for classification and regression. SVMs transform the attribute space into a higher dimensionality space called feature space where the classes of the domain can be separated linearly by an hyperplane. This specific implementation is a simplification of both *SMO* (Platt, 1999) and *SVMLight* (Joachims, 2002).

Majority class As a baseline, a classifier that always predicts the majority class existing in the training set is also included.

XCS (Wilson, 1995) This is the most popular system of the Michigan approach of GBML. We selected version of XCS is XCSTS (Butz, Sastry, & Goldberg, 2003), which used tournament selection instead of the usual fitness-proportionate one.

With the exception of *LIBSVM* and *XCS*, we have used the *WEKA* (Witten & Frank, 2000) implementation of these algorithms, using the default parameters. For the *IBk* system we have used a k of 3, and for the *NaiveBayes* the two kinds of approaches to deal with real-valued attributes (gaussian and non-parametric kernel density) have been tested.

The *XCS* results were provided by Martin Butz, and used different sets of cross-validation partitions. Therefore, the *XCS* results will not be included in the statistical tests. Also, the results for some datasets are missing. Thus, when the average results are computed, they will only include the datasets for which *XCS* has results. The settings of this system appear in (Bacardit & Butz, 2004).

9.2 Configurations of GAssist included in the comparison

In each of the four previous chapters it was decided, for the specific topic of study of the chapter, which was the best technique in the framework of GAssist. In this chapter we will not experiment again with all the studied techniques, but use only the best method of each chapter.

However, in the chapter dealing with the *ADI* knowledge representation there was no clear winner. The alternative instances set/1-NN representation had slightly better performance, but also much higher computational cost. Therefore, both techniques will be included in this global comparison. Also, in all chapters beside the specific chapter dealing with *ADI*, the set of discretization algorithms used for *ADI* was the set of uniform-width discretizers used in previous work, instead of the set with a mix of supervised and unsupervised discretizers that the experimentation in this thesis determined to be the best.

Thus, in this global comparison three sets of discretizers will be tested. The first one is the old uniform-width set of discretizers and the other two are the ones that obtained better accuracy in the *ADI* experimentation. Why include three configurations of *ADI*? As said in the conclusions of chapter 6, these sets of discretizers were determined to be the best for *ADI* using 15 datasets. 33 datasets (27 of them with real-valued attributes) will be included in the experimentation of this chapter. Thus, it is an opportunity to validate partially the performance of these groups of discretizers.

Table 9.1: Settings of GAssist for the global comparison with other machine learning systems

Parameter	Value
General parameters	
Crossover prob.	0.6
Selection algorithm	tournament selection
Tournament size	3
Population size	400
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Iterations	A maximum of 3000
Minimum number of rules for fitness penalty	maximum of 6
Default class policy	auto
Number of strata of ILAS windowing	2
ADI knowledge representation	
Probability of ONE	0.75
Probability of Split	0.05
Probability of Merge	0.05
Maximum number of intervals	5
Rule deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes in dataset + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ratio	0.075
Weight relax factor	0.90

For the domains containing only nominal attributes the *GABIL* representation has been used. Thus, all configuration of *GAssist* included in the comparison will show the same results for these datasets. Table 9.1 contains the configuration of *GAssist* for the experimentation described in this chapter, and table 9.2 describes the sets of discretizers used.

9.3 Global comparison experimentation

The experimentation will be split into two parts. The first part will deal with the all the small datasets described in table 4.1 of chapter 4. The second part will be focused on the medium-size datasets described in table 4.2 of the same chapter.

Table 9.2: Groups of discretizers used by ADI in the global comparison

	Group 1	Group 2	Group 3
Initial prob. of reinit.	0.04	0.03	0.03
Discretization algs.	Màntaras	Màntaras	Uniform-width 4
	MDLP	MDLP	Uniform-width 5
	Uniform-width 5	Uniform-width 5	Uniform-width 6
	Uniform-width 20	Uniform-width 20	Uniform-width 7
	Uniform-width 10		Uniform-width 8
	Uniform-width 15		Uniform-width 10
	Uniform-frequency 10		Uniform-width 15
	Uniform-width 25		Uniform-width 20
			Uniform-width 25

9.3.1 Experimentation on small datasets

For this comparison we are interested in reporting two kinds of results: test performance (obviously) and complexity of the generated solutions. Table 9.3 contains the average results of training accuracy, test accuracy and size of the solutions for all the learning systems and configurations of *GAssist* included in the comparison. The full results of these tests appear in appendix D.

The size of the solutions (being either number of rules, number of leaves or number of support vectors) has been extracted from all the learning systems where such measure is available: *GAssist*, *C4.5*, *PART* and *LIBSVM*. The population size of *XCS* was also available, but it would be unfair to use it as the size of the solution because it contains rules that are never used in the exploitation stage. In these tables, the learning systems are identified as follows:

- GAssist with ADI representation and group 1 of discretizers - **GAssist-gr1**
- GAssist with ADI representation and group 2 of discretizers - **GAssist-gr2**
- GAssist with ADI representation and group 3 of discretizers - **GAssist-gr3**
- GAssist with instance set representation - **GAssist-inst**
- Majority class classifier - **Majority**
- C4.5 - **C4.5**
- PART - **PART**

Table 9.3: Average results of the global comparison tests on small datasets

System	Training accuracy	Test accuracy	Complexity of solution
GAssist-gr1	89.39±10.99	80.97±13.03	7.40±3.76
GAssist-gr2	89.16±11.05	80.90±12.98	7.29±3.75
GAssist-gr3	89.83±10.97	80.95±13.07	7.76±3.79
GAssist-inst	90.89±10.70	81.39±13.14	17.53±11.81
Majority	52.27±16.87	52.25±16.89	
C4.5	90.63±8.63	79.33±12.28	27.46±30.28
PART	91.23±8.43	79.47±12.59	21.53±30.41
IB1	99.07±2.48	78.59±14.37	
IBk	88.34±8.89	80.21±13.55	
NB-gaussian	80.34±14.75	77.88±15.86	
NB-kernel	83.23±12.57	79.72±14.40	
LIBSVM	83.53±13.85	79.74±15.08	237.23±249.76
XCS	95.28±10.33	79.66±12.78	

- IB1 - **IB1**
- IBk with $k = 3$ - **IBk**
- NaiveBayes with gaussian estimator for real-valued attributes - **NB-gaussian**
- NaiveBayes with kernel density estimator for real-valued attributes - **NB-kernel**
- LIBSVM Support Vector Machine library - **LIBSVM**
- XCS - **XCS**

Looking at the average test accuracy of the learning systems included in the comparison we can see how all the configurations of *GAssist* have the top results of the table. However, this comparison is somewhat unfair for one reason: the method used to handle missing values. *GAssist* uses a substitution policy, detailed in chapter 4 while most of the other methods use a simpler policy of ignoring the missing values (as if that value was irrelevant) in the match process. These global averages are split in two tables; table 9.4 containing the averages of the datasets where more than 10% of the instances have missing values, and table 9.5 with the rest of datasets.

The average results on the datasets with missing values show how the policy used in *GAssist* seems much better than the policies used in the rest of systems. The average accuracy difference between the worst configuration of *GAssist* (GAssist-inst) and the best of the other learning systems (PART) is 2.65%. Moreover, the statistical t-tests applied to these results, described in table 9.6, confirm this observation because the configurations of *GAssist* are the

Table 9.4: Average results of the global comparison tests on small datasets with more than 10% of instances with missing values

System	Training accuracy	Test accuracy	Complexity of solution
gr1	89.78±13.03	82.69±16.21	9.36±5.22
gr2	89.81±12.94	82.73±15.96	9.45±5.18
gr3	89.91±13.03	82.81±16.08	9.42±5.20
inst	90.93±12.58	81.68±15.80	15.25±9.67
ZeroR	47.64±22.14	47.60±22.07	
C4.5	88.03±10.52	78.94±14.58	23.67±20.64
PART	89.09±9.96	79.03±14.43	17.08±12.78
IB1	97.78±3.97	76.47±16.09	
IBk	86.03±11.84	78.51±15.05	
NB-gaussian	81.76±12.21	78.10±15.02	
NB-kernel	82.62±11.21	78.73±14.29	
svm	84.75±12.48	78.68±16.08	190.85±143.14
xcs	89.95±15.11	77.49±14.69	

Table 9.5: Average results of the global comparison tests on small datasets with less than 10% of instances with missing values

System	Training accuracy	Test accuracy	Complexity of solution
GAssist-gr1	88.99±9.81	79.61±11.42	6.55±2.43
GAssist-gr2	88.64±9.94	79.54±11.42	6.36±2.34
GAssist-gr3	89.54±9.79	79.62±11.44	7.03±2.62
GAssist-inst	90.36±9.84	80.50±12.08	18.06±12.37
Majority	54.34±13.24	54.32±13.32	
C4.5	91.23±7.59	78.70±11.48	28.17±32.95
PART	91.41±8.09	78.86±12.01	22.56±34.63
IB1	99.64±0.99	78.71±13.64	
IBk	88.98±7.06	80.32±12.80	
NB-gaussian	79.11±15.57	77.21±16.06	
NB-kernel	82.69±13.33	79.45±14.45	
LIBSVM	82.29±14.42	79.52±14.60	252.14±275.57
XCS	97.57±6.10	80.59±11.75	

Table 9.6: T-tests applied over the results of the global comparison in small datasets with missing values, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	GAssist-gr1	GAssist-gr2	GAssist-gr3	GAssist-inst	Majority	C4.5	PART	IB1	IBk	NB-gaussian	NB-kernel	LIBSVM	Times outperforming
GAssist-gr1	-	0	0	2	9	5	6	6	6	6	6	6	52
GAssist-gr2	0	-	0	2	9	5	6	6	5	5	5	5	48
GAssist-gr3	0	0	-	2	9	5	6	6	5	5	6	6	50
GAssist-inst	0	0	0	-	9	5	5	6	5	5	5	5	45
Majority	0	0	0	0	-	0	0	0	0	0	0	0	0
C4.5	3	3	3	3	8	-	2	6	4	4	4	3	43
PART	3	3	3	3	8	2	-	6	4	4	4	2	42
IB1	3	3	3	3	8	0	1	-	2	3	3	2	31
IBk	1	1	1	2	8	4	2	5	-	3	3	1	31
NB-gaussian	1	1	1	1	9	5	5	5	4	-	0	3	35
NB-kernel	1	1	1	1	9	5	5	5	4	1	-	4	37
LIBSVM	1	1	1	1	9	5	5	7	5	3	3	-	41
Times outperformed	13	13	13	20	95	41	43	58	44	78	39	37	

ones significantly outperforming the other learning systems more times than any other system, and are also the ones being outperformed least.

The results on the datasets with few/no missing values give an slightly different picture. In these results the configurations of *GAssist* still have a competent performance, especially *GAssist-inst*, but they are not the top performing methods in general. The t-tests applied to these results, summarized in table 9.7, confirm these observations. Leaving *XCS* aside because it could not be included in the t-tests, the best learning system is *LIBSVM* because it is the second system being outperformed least, thus it is very robust, and also the one outperforming the other systems most. The configurations of *GAssist* are also quite robust, being outperformed few times, although they are not the at the top of the outperforming count ranking.

So far the comparison has mixed all kinds of knowledge representations, but it would be also interesting to split it into orthogonal/non-orthogonal ones. Tables 9.8 and 9.9 contain the average results and t-tests respectively for orthogonal knowledge representations, and tables 9.10 and 9.11 the average results and t-tests respectively for the non-orthogonal ones. These results belong to the datasets without missing values.

Table 9.7: T-tests applied over the results of the global comparison in small datasets without missing values, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	GAssist-gr1	GAssist-gr2	GAssist-gr3	GAssist-inst	Majority	C4.5	PART	IB1	IBk	NB-gaussian	NB-kernel	LIBSVM	Times outperforming
GAssist-gr1	-	0	0	4	22	10	11	11	5	10	7	5	85
GAssist-gr2	0	-	0	4	22	10	12	10	5	10	7	5	85
GAssist-gr3	0	0	-	4	22	10	11	9	3	10	8	5	82
GAssist-inst	6	8	7	-	22	10	9	10	8	10	7	5	102
Majority	0	0	0	0	-	1	2	1	1	1	1	0	7
C4.5	2	2	2	3	22	-	3	8	5	10	4	6	67
PART	3	3	3	1	21	5	-	9	5	9	5	5	69
IB1	5	6	6	2	21	8	10	-	3	9	6	5	81
IBk	10	10	10	5	23	11	11	13	-	10	7	4	114
NB-gaussian	8	9	7	6	21	11	12	8	7	-	1	4	94
NB-kernel	9	10	8	8	22	13	11	11	11	10	-	8	121
LIBSVM	14	15	14	11	21	16	16	18	11	14	9	-	159
Times outperformed	57	63	57	48	239	105	108	108	64	165	62	52	

Table 9.8: Average results of the global comparison tests on small datasets with less than 10% of instances with missing values for the orthogonal knowledge representations

System	Training accuracy	Test accuracy	Complexity of solution
GAssist-gr1	88.99±9.81	79.61±11.42	6.55±2.43
GAssist-gr2	88.64±9.94	79.54±11.42	6.36±2.34
GAssist-gr3	89.54±9.79	79.62±11.44	7.03±2.62
C4.5	91.23±7.59	78.70±11.48	28.17±32.95
PART	91.41±8.09	78.86±12.01	22.56±34.63
XCS	97.57±6.10	80.59±11.75	

Table 9.9: T-tests applied over the results of the global comparison in small datasets without missing values and for orthogonal knowledge representations, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	GAssist-gr1	GAssist-gr2	GAssist-gr3	C4.5	PART	Times outperforming
GAssist-gr1	-	0	0	10	11	21
GAssist-gr2	0	-	0	10	12	22
GAssist-gr3	0	0	-	10	11	21
C4.5	2	2	2	-	3	9
PART	3	3	3	5	-	14
Times outperformed	5	5	5	35	37	

These results show how the evolutionary methods, *XCS* and *GAssist* have the best test performance. Lacking the statistical tests over *XCS*, the configurations of *GAssist* have the best scores on the t-tests, showing robustness and good performance. Also, the solutions generated are also the smallest, thus probably more interpretable which is a very interesting characteristic of a learning system.

Table 9.10: Average results of the global comparison tests on small datasets with less than 10% of instances with missing values for the non-orthogonal knowledge representations

System	Training accuracy	Test accuracy	Complexity of solution
GAssist-inst	90.36±9.84	80.50±12.08	18.06±12.37
IB1	99.64±0.99	78.71±13.64	
IBk	88.98±7.06	80.32±12.80	
NB-gaussian	79.11±15.57	77.21±16.06	
NB-kernel	82.69±13.33	79.45±14.45	
LIBSVM	82.29±14.42	79.52±14.60	252.14±275.57

Table 9.11: T-tests applied over the results of the global comparison in small datasets without missing values and for non-orthogonal knowledge representations, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	GAssist-inst	IB1	IBk	NB-gaussian	NB-kernel	LIBSVM	Times outperforming
GAssist-inst	-	10	8	10	7	5	40
IB1	2	-	3	9	6	5	25
IBk	5	13	-	10	7	4	39
NB-gaussian	6	8	7	-	1	4	26
NB-kernel	8	11	11	10	-	8	48
LIBSVM	11	18	11	14	9	-	63
Times outperformed	32	60	40	53	30	26	

The comparison of the non-orthogonal knowledge representations reflects the same trends as the overall comparison. Although *GAssist-inst* and *IBk* have the best test averages, the t-tests indicate that *LIBSVM* has the best performance, followed by *NB-kernel* and *GAssist-inst* with similar results. With only two of the methods including results about the complexity of the solutions generated, conclusions about this issue cannot be extracted.

Table 9.12: Average results of the global comparison tests on large datasets

System	Training acc.	Test acc.	Size of sol.	Time (s)
GAssist	87.39±9.93	86.91±10.05	10.99±4.88	300.00±0.00
C4.5	95.84±5.35	91.61±8.26	942.57±1839.83	8.51±14.45
NB-kernel	87.66±7.13	87.24±7.20		8.89±13.00
PART	96.32±4.65	91.58±8.41	761.70±1439.81	425.53±1037.80

9.3.2 Experimentation on large datasets

For the experimentation on large datasets we reproduce the results reported in chapter 7, selecting the configuration with best training accuracy for each dataset with run-time 300, as the tests showed that is a good strategy. That test used *ADI* representation and group 3 of discretization algorithms. No extra tests will be done with other settings of *ADI* because the results on small datasets showed that the existing differences between the groups of discretizers are minor when compared to the differences with other learning systems. This time the *IB1*, *IBk* and *LIBSVM* methods are not included in the comparison because their run-time is completely excessive, using more than an hour per run (fold) in some datasets. *XCS* is also removed because results are missing for half of the datasets used, so it is difficult to extract proper conclusions. Finally, as the *NB-kernel* configuration of NaiveBayes showed in the previous tests to be much better than *NB-gaussian*, it is the only one included.

Table 9.12 contains the average results over all datasets of these tests. As usual, the full detail of these results is in appendix D. These results were analyzed with statistical t-tests, that are summarized in table 9.13. Clearly, *GAssist* is not prepared to handle these datasets in a reasonable time (let us remind that a time limit of 300 seconds was applied to the *GAssist* runs). The same statement can be made for the NaiveBayes system. Nevertheless, the small size of the solutions generated, compared to *C4.5* or *PART* is a very good feature of this system. Even if these solutions are not as accurate as the ones of the other system, the human experts will probably extract more useful information due to its small size.

9.4 Analysis of the experimentation results

In this section we will analyze the results reported in the previous section from many different points of view, and we will try to explain why *GAssist* performs as it does. Most of the chapter will be focused on small datasets, because the conclusions extracted from these datasets are also applicable to the larger ones.

Table 9.13: T-tests applied over the results of the global comparison in big datasets, using a confidence level of 0.05. Cells in table count how many times the method in the row significantly outperforms the method in the column.

	GAssist	C4.5	NB-kernel	PART	Times outperforming
GAssist	-	0	5	0	5
C4.5	12	-	11	5	28
NB-kernel	7	2	-	3	12
PART	12	4	10	-	26
Times outperformed	31	6	26	8	

9.4.1 The handling of missing values

The results of tests reported in the previous section showed why it is crucial to correctly choose the policy used to handle missing values. In this sense, *GAssist* showed to have chosen its policy (of substitution) more correctly than the other systems. Moreover, the statistical tests showed that *GAssist* managed to outperform the other systems in 5-6 datasets but it was outperformed 3 times by *C4.5*, *PART* and *IB1*.

Looking at the specific datasets of these significant differences, we can see that all the dataset where *GAssist* was significantly better have something in common: there is a majority of nominal attributes. On the other hand, we can see two different patterns in the three datasets where *GAssist* was outperformed. On one hand we have the only dataset with missing values where real-valued attributes are majority, and on the other hand we have two datasets with 19 and 24 classes.

Although we only have data on 9 datasets, it seems that the substitution policy for nominal attributes is much better than the policy for real-valued ones. This in an issue that needs further work to be able to extract strong conclusions. Also, it seem that *GAssist* has problems with datasets with a high number of classes. This last issue will appear again later in this section, and it is one of the areas where *GAssist* can be improved in the future.

9.4.2 Generalization capacity of the compared systems

Comparing the averages of training and test accuracies for the small datasets, in table 9.5, we can see three kinds of behaviors. First of all we have three systems (*NB-gaussian*, *NB-kernel* and *LIBSVM*) where the difference between the achieved training and test accuracy is quite small (less or equal than 3%). Then, we have the 3 configurations of *GAssist* using

ADI representation, with a medium level of difference. Finally, *GAssist-inst*, *C4.5*, *PART* and especially *XCS* have the highest difference between training and test accuracies. In this analysis we have left *IB1* and *IBk* aside because, strictly speaking, it cannot be said that they learn.

We can observe two totally opposite behaviors. First there are *NB-gaussian*, *NB-kernel* and *LIBSVM*, that show not to have almost any over-learning, but that sometimes cannot learn all the correct knowledge. On the opposite end there is *XCS*, showing that it suffers from a high degree of over-learning but also showing the higher test accuracy, thus showing that it is able to learn more correct knowledge than the other systems. In the middle there are the other systems, especially the *ADI* configurations of *GAssist*. They suffer from more over-learning than *NaiveBayes* and *LIBSVM* but they are able to learn more correct knowledge than these methods, showing higher (although the difference is small) test accuracy.

9.4.3 The size of the generated solutions

If we look at the systems that provide a complexity measure of the generated solutions, the configurations of *GAssist* (especially the *ADI* ones) provide the solutions of smaller size. This is a very good feature of any learning system, and clearly is one of the strongest favorable points of *GAssist*.

Although we do not have exact solution size results from *XCS* they are probably bigger than the solutions generated by *GAssist* for two reasons: first of all, it is very difficult to achieve so high a training accuracy (an average of 97%) with a reduced set of rules. Second, even if we can filter the population of *XCS* to extract a reduced set of rules, the final number of rules probably will be still quite high compared to *GAssist*. To cite an example of a rule reduction algorithm for *XCS* (Wilson, 2002), a reduction of a population of around 1200 rules to 25 final rules for the *wbcd* dataset was achieved. However, *GAssist* generates only 3 rules for this domain. Therefore, the difference is still large.

The *NaiveBayes* system creates a probability distribution table for each combination of attribute and class of the problem. Therefore, the solution size would be still quite big. Finally, the *IB* methods use all the training set as the core for its classifier method. Thus, they are completely out of this comparison.

9.4.4 What datasets are hard for GAssist and ADI?

The general observation that can be extracted from the performance of *GAssist* on small datasets is that it has quite good performance, although not the best one. Can we extract some common pattern from the the results of the datasets where *GAssist* and, specifically,

Table 9.14: Performance of *GAssist-gr3* and *XCS* in the small datasets where *XCS* has better performance

Dataset	System	Training acc.	Test acc.	Size of sol.
bal	GAssist-gr3	86.34±0.65	78.66±3.86	11.12±2.24
	XCS	95.19±1.28	81.1±3.8	
bpa	GAssist-gr3	82.20±1.54	63.29±8.69	8.51±1.44
	XCS	99.97±0.10	67.1±7.5	
gls	GAssist-gr3	80.64±1.93	68.30±9.27	6.85±1.04
	XCS	97.62±1.37	71.8±8.9	
son	GAssist-gr3	97.03±1.16	77.21±8.86	7.20±1.21
	XCS	100.00±0.00	77.9±8.0	
thy	GAssist-gr3	98.46±0.67	92.13±5.30	5.48±0.65
	XCS	99.90±0.44	95.4±4.6	
veh	GAssist-gr3	72.62±1.11	67.87±3.42	8.08±1.66
	XCS	98.68±0.62	74.3±4.7	
wdbc	GAssist-gr3	98.31±0.44	93.82±2.94	4.72±0.82
	XCS	100.00±0.00	96.0±2.5	
wine	GAssist-gr3	99.51±0.50	93.83±5.43	3.73±0.71
	XCS	100.00±0.00	95.6±4.9	
zoo	GAssist-gr3	98.06±1.26	91.28±8.31	7.44±0.72
	XCS	100.00±0.00	95.1±6.1	
Ave.	GAssist-gr3	90.35±9.48	80.71±11.67	7.01±2.08
	XCS	99.04±1.57	83.81±11.08	

the *ADI* representation does not perform well? Table 9.14 compares the performance of *GAssist-gr3* and *XCS* in the datasets where *XCS*, the best system using orthogonal knowledge representation, has much better performance.

From the performance of both systems on these datasets, and the features of the dataset we can extract some patterns. First of all, the average training accuracy of *XCS* on these datasets is is 1.5% higher than the global average on datasets without missing values, while the increase of *GAssist* is only of 0.81%. Moreover, 6 of these 9 datasets have more than 2 classes, and some of these datasets have a very uneven class distribution. These observations indicate a problem of *GAssist*: That it is unable to learn rules that cover few examples when compared to the average coverage of the rule set. on the other hand, XCS is able to learn rules covering almost all the training set, something reflected by its high training accuracy.

This problem is well illustrated showing a rule-set generated for the *bal* dataset where *GAssist* was unable to induce rules covering the minority class B (having 7.84% of the instances):

Table 9.15: Performance of *GAssist-gr3* and *XCS* in the small datasets where *GAssist* has better performance

Dataset	System	Training acc.	Test acc.	Size of sol.
cmc	GAssist-gr3	59.59±1.12	54.74±4.05	10.31±3.03
	XCS	71.22±2.34	52.4±3.6	
cr-g	GAssist-gr3	82.74±0.92	72.29±4.13	12.69±2.02
	XCS	97.92±0.81	70.9±4.3	
h-c	GAssist-gr3	93.70±0.76	80.28±6.29	8.08±1.49
	XCS	99.81±0.31	76.5±7.9	
h-s	GAssist-gr3	92.75±0.87	80.27±8.11	7.45±1.12
	XCS	99.85±0.24	75.3±8.1	
ion	GAssist-gr3	96.90±0.74	92.71±5.01	2.59±0.93
	XCS	99.86±0.24	90.1±4.7	
lym	GAssist-gr3	95.89±1.32	80.80±10.82	6.69±1.11
	XCS	98.84±0.84	79.8±10.2	
pim	GAssist-gr3	83.72±0.81	74.36±5.05	8.71±1.81
	XCS	98.90±0.67	72.4±5.3	
Ave.	GAssist-gr3	86.47±12.15	76.49±10.73	8.07±2.91
	XCS	95.20±9.81	73.91±10.55	

0:Att 0 is [2.33333,5],Att 1 is [1.2,5],Att 3 is [1,2.76] → L
1:Att 0 is [1.26667,5],Att 1 is [1,4.2][4.6,4.8],Att 2 is [1,4],Att 3 is [1,1.6] → L
2:Att 0 is [1.26667,5],Att 1 is [2.2,5],Att 2 is [1,3],Att 3 is [1,4.8] → L
3:Att 0 is [4,5],Att 1 is [4,5],Att 3 is [1.4,5] →L
4:Att 0 is [1,3.66667][4.33333,5],Att 1 is [3,5],Att 2 is [1,4.73333],Att 3 is [1,1.16] → L
5:Att 0 is [1.26667,5],Att 2 is [1,4.73333],Att 3 is [1,1.96][2.12,2.6] →L
6:Att 1 is [1.57143,5],Att 2 is [1,1.8] →L
7:Att 0 is [2.33333,5],Att 1 is [2.2,5],Att 2 is [1,4] → L
8:Default rule → R

9.4.5 What datasets are easy for GAssist?

The aim of this subsection is the opposite of the previous one: we want to compare again the performance of *GAssist* and *XCS* but now in the datasets (leaving missing values aside) where *GAssist* is better. Table 9.15 contains the results for these datasets.

Looking at the average training accuracy of *GAssist* and *XCS* on these datasets we see the opposite behavior to the one showed by the results on the datasets in the previous subsection. Both systems suffer a training accuracy drop compared to the global average in table 9.5, with *GAssist* having the largest drop. Moreover, most of these datasets have only two classes. Also, in the three datasets where the test accuracy difference is larger (*h-c*, *h-s* and *ion*), GAssist

selected the minority class for the default rule using the automatic determination of default class policy introduced in chapter 5.

The conclusion that emerges from these patterns is that the over-learning problem of *XCS* observed previously is amplified in these datasets because it cannot generate rules with almost perfect training accuracy, thus it is unable to learn all the correct knowledge of the dataset. On the other hand, *GAssist* is able to avoid the over-learning problem thanks to the explicit default rule mechanism for two reasons: (1) the search space reduction due to not using the default class in the rest of rules is maximum in the datasets with few classes and (2) choosing the correct class hierarchy by selecting the minority class for the default rule usually creates more compact (thus better generalized) solutions.

9.4.6 Performance of GAssist on large datasets

The results of the experimentation on large datasets showed in general that *GAssist* has a bad performance level. What is the reason for this? First of all, it is necessary to remember the time limit of 300 seconds used in *GAssist*. Probably with some more time its performance would be better. The question is if it is worth using a system that needs much more learning time. A learning period slightly longer would still be relatively reasonable, but the real solution would be to know **why** it is learning so slowly.

The answer to this question can be extracted partially from these results, because the datasets where the performance of *GAssist* has been significantly outperformed are the ones with more classes. Thus, the problem is the same one that was identified for the small datasets, but much more amplified. Also, it is possible that because of the windowing process, some minority classes are not represented enough in all the strata and therefore cannot be learned.

9.5 Conclusions and further work

After reporting an extensive experimentation to compare the performance of *GAssist* with other learning systems, the general conclusions that we can extract from these tests is that the contributions presented in this thesis improve the performance of the *Pittsburgh* model, and create a system that has competent performance compared to a broad range of learning paradigms. Moreover, the two kinds of knowledge representations used in *GAssist* that were included in this comparison perform well within their respective groups of orthogonal/non-orthogonal representations.

However, the experimentation identified some weak points of *GAssist* that need to be improved. Most problems detected can be reduced to a single issue: learning low-frequency rules. It is very difficult for this system to induce rules for an individual that cover very few examples compared to the rest of the individual rules. This problem is especially critical for large datasets.

The cause of this problem are the generalization pressure operators of the system, but probably the system would be able to learn some more rules if the parameters that control each of the studied generalization pressure operators (the hierarchical selection and the MDL-based fitness function) were relaxed. However, this relaxation could be bad for the overall performance of the system for two reasons: (1) the average number of rules generated on most datasets would be increased. Right now one of the most positive features of *GAssist* is that it generates few rules. Is it worth partially losing this capacity? and (2) probably the test performance of the system on some datasets would be decreased because of over-learning.

Therefore, alternative methods must be studied. Among other options, there are two to look into:

- Adding some kind of covering operator. The cause of the high training accuracy of *XCS* is partially due to an operator, called covering, that deterministically introduces rules into the population if there are no individuals that are able to match an input instance. *GAssist*, by lacking such operator, must generate these rules with the classical blind genetic operators. If the rule covers few examples it has a really small probability of surviving in the population because it is almost impossible to create such rule with a single genetic operation. This problem would disappear if we could deterministically add the correct rules to the correct individuals. Obviously an operator of this kind would increase the size of the generated rule set. The question, as usual, is to find the proper accuracy-complexity balance.

- Adapting the methods used in the machine learning community (Japkowicz & Stephen, 2002). Usually two kinds of methods are reported in the literature:

  - Sub-sample the majority class/super-sample the minority class. This approach is very interesting for our system because it is very close to what the *ILAS* windowing schema is already using. Also, if we are able to use properly the stratification success model developed in chapter *windowing*, a theoretical model for this issue could be developed.
  - Changing the cost of miss-classifying the examples of different classes based on the class distribution. This issue would mean, in the context of a *Pittsburgh* model, changing the fitness function, which is something relatively easy. However, this is

only half of the task: the recombination operators of the *GA* still have to generate the rules covering the minority class.

Therefore, it seems that this issue can potentially be solved with minor changes to *GAssist*. If this is true, it will be the final validation of the contributions presented in this thesis.

9.6 Summary of the chapter

This chapter has reported the experiments that were lacking in the previous chapters, where each of the contributions presented in this thesis were compared only to some alternatives inside the framework of *GAssist*. The experimentation of this chapter compared seven machine learning systems with *GAssist*, using all the contributions presented in this thesis.

The experiments revealed several issues. First of all, the policies for handling missing values used in *GAssist* are more appropriate than the policies used in the rest of systems. Leaving this issue aside because it is not strictly related to learning (it is only a pre-processing stage), *GAssist* showed a very competent performance compared to the alternative learning systems. Moreover, the different knowledge representations of *GAssist* perform well in their respective kinds of knowledge representations.

Although the results so far validated highly the contributions presented in this thesis, we were also interested in detecting if the system has some strong and weak points, and why. The comparison with *XCS*, the system with orthogonal knowledge representation that had the best test results, reflected the major problem of *GAssist*: learning rules that cover few examples of the training set. This problem also appeared in the experimentation with large datasets, but even more amplified.

As further work, some solutions were proposed for this problem. Nevertheless, this issue does not invalidate the contributions presented in this thesis because most of these contributions are not affected directly by this problem.

Part III

Conclusions and further work

Chapter 10

Conclusions

In this thesis several kinds of contributions have been presented, tested first in a reduced comparison framework and later in a larger-scale comparison with many alternative machine learning systems. Now it is time to extract conclusions from all the research reported in this thesis. First we will summarize the work done and the contributions extracted for each of the four kind of contributions presented in this thesis. After, some general conclusions will be presented.

10.1 Conclusions about the explicit default rule mechanisms

In chapter 5 we studied some mechanisms that explicitly exploit a certain emergent feature that appears if the rule-set is used as a decision list: the automatic generation of a default rule. The initial approach studied was very simple: extend the knowledge representation with an explicit and static default rule. This means that the class used for this default rule is not used in the "dynamic" rules of the individual, effectively reducing the search space and potentially reducing the danger of over-learning, because it generates more compact rule sets.

However, the experiments done discovered that the choice of the class assigned to this default rule is not trivial. Simple policies such as using the majority/minority class as the default class perform quite well compared to the original system. However, they perform poorly on certain datasets that somewhat show a lack of robustness. We can almost integrate the best results of both policies by using the simple heuristic of selecting the policy with more training accuracy. This mechanism introduces a good performance boost, but doubles the run-time.

As an alternative, a mechanism that can determine automatically the default class was developed. This technique works by integrating in a single population individuals using all possible default classes, and letting them compete among themselves. A niching mechanism is introduced to guarantee that all the niches (default classes) survive in the population enough time to learn properly and assure that a fair comparison between the default classes is achieved. This automatic mechanism performed better when we increased the population size, which is a usual requirement in most systems that use niching. Nevertheless, it showed almost similar performance to the policy mentioned above of combining the majority/minority policies while having significantly less run-time.

All the policies for the determination of the default class managed to outperform the version of the system without explicit default class, validating the mechanism. Moreover, for the reasons stated above we think that the automatic policy is the best method.

The comparison of *GAssist* with *XCS* of the previous chapter showed that the domains where *GAssist* is much better than the other system are, indeed, the datasets where this default class mechanism can be potentially more beneficial: the common behavior of *XCS* on these datasets was that it was not able to achieve the usual level of training accuracy, reflecting that it had problems creating a model for training set. It is in these "tough" datasets where the search space reduction introduced by the explicit default rule mechanism becomes crucial to learn properly, validating the benefits that it introduces.

10.2 Conclusions about the *ADI* knowledge representation

The second contribution presented in this thesis, in chapter 6 was a knowledge representation for real-valued attributes called *Adaptive Discretization Intervals (ADI) knowledge representation*. This representation handles real-valued by means of discretization algorithms, thus being able to reuse well-known nominal representations. This process usually has two pitfalls: (1) sometimes, especially with non-supervised discretizers, not all generated cut-points are meaningful therefore wasting exploration power on useless areas of the search space and (2) dealing with the bias introduced by the discretization algorithm. Does the dimensionality reduction introduced by a discretization process is adequate for all datasets? The answer most times is no.

ADI solves the first problem by using *adaptive* intervals. These intervals are constructed over the cut-points generated by the discretization algorithm. Therefore, if the cut-point splitting two intervals is irrelevant, the representation will eliminate it *merging* the intervals.

The second problem is solved by using several discretization algorithms at the same time, thus automatically using for each dataset the most suitable discretization.

The experimentation showed that the approach of combining multiple discretization algorithms is feasible because all the tested combinations of discretizers achieved more average accuracy than the best single discretization algorithm. Also, inside the framework of *GAssist* this representation had better performance that the representation evolving real-valued intervals. Initially it looked like *ADI* were better at exploring properly the search space but the experimentation on generalization pressure methods (GPM) in chapter 8 brought more light to the issue: the studied GPM methods (especially the hierarchical selection operator) were unable to avoid completely the introduction of some small over-learning in the generated rule sets. In *ADI* this problem did not exist because the discretization process eliminates it partially. Thus, it can be said that *ADI*, although not explicitly, also introduces some generalization pressure.

The comparison of *ADI* with *XCS*, *C4.5* and *PART* showed that it has competent performance compared to these systems with orthogonal knowledge representations. *XCS* was the only system that was better than *GAssist+ADI*, and as was discussed in the previous section, probably the cause of this fact is not strictly related to the knowledge representation, but to other parts of the system like the *covering* operator of *XCS*. Therefore, the global comparison validated that *ADI* is a competent orthogonal knowledge representation.

10.3 Conclusions about the windowing mechanisms

Chapter 7 mainly focused on a kind of windowing mechanism, called *ILAS* (Incremental Learning with Alternating Strata), that showed some interesting characteristics: beside the obvious run-time reduction, it introduced extra generalization pressure to the learning process.

The rest of the chapter focused on exploiting these two characteristics: determining the maximum run-time reduction possible without significant impact in the performance or maximizing the performance of the system, independently of the run time (for small datasets). In order to achieve these objectives two models were developed. One about the maximum number of strata that can be used without degrading the performance of the system. By this we mean computing the probability that the created strata are still enough representative of the whole training set. The other one about the run-time of *ILAS*. Both models were developed using synthetic (and thus, predictable) datasets.

When these two models were put into practice using real datasets, the experimentation showed that the stratification representativity model needs to be refined if it has to predict the number of strata that gives maximum test accuracy. Some ideas of how to do this

refinement were proposed, that are to be left for further work. The experimentation with real datasets showed that the run-time model needed to be expanded. The expanded model was more accurate in small datasets, but it was not usable in large datasets, because of physical limitations of the experimentation framework.

Nevertheless, the experimentation showed that we can still use deterministic strategies easy to tune to achieve these objectives and for small and large datasets that in general have good performance. *ILAS* can improve the performance of the Pittsburgh model of GBML in more than one way. The tests showed how *ILAS* can be used successfully in several different scenarios, showing its versatility.

However, probably *ILAS* is partially the cause of the problem that *GAssist* suffers of being unable to learn low-frequency rules properly, a problem observed in the global comparison in previous chapter.

10.4 Conclusions about the bloat control and generalization pressure methods

Chapter 8 focused on solving two very closely related problems. The first problem is a common issue in evolutionary computation techniques that use variable-length representations: the bloat effect. The way to control this problem in the framework of *GAssist* is a rule deletion operator that, properly controlled, can be beneficial in two aspects: run-time reduction and introduction of diversity.

The second problem is related to the machine learning field: the capacity of the learning system to generate well generalized solutions. Usually a well generalized solution is identified as an accurate solution of low complexity. Thus, the explicit control of the generalization issue is also closely tied to the control of the individuals size. Two alternative methods have been proposed in this thesis to apply generalization pressure in the system. The first one, the hierarchical selection operator, is very simple. The second one, the MDL-based fitness function, is much more complex.

The experimentation performed showed that the MDL-based fitness function was the best method in different aspects: It was the one achieving the test accuracy and it was the one working well in different knowledge representations with the same set of parameters. These experiments have to be interpreted with a grain of salt until we can determine if the other two methods included in the comparison can be tuned properly, because all three methods were tested under the same circumstances, which maybe are unfair for some methods.

Nevertheless, even if other methods can achieve similar performance than the MDL-based fitness function, this method is still relevant because of its novelty: it is one of the few methods in this area that considers the content of the individuals to guide the exploration process, unlike most methods that only take into account its behavior (training accuracy) and very simple measures of complexity.

10.5 Global conclusions of the thesis

The global conclusions that can be extracted from the thesis are positive. All the four kinds of contributions studied are able to improve the basic *Pittsburgh* model and, even more important, can be combined among themselves. The performance of the final *GAssist* version, containing the best set of the studied techniques, is very competent compared to several kinds of machine learning systems. Also, *GAssist* generates solutions of very reduced size, which is a very important feature of a learning system from an interpretability point of view.

However, the global comparison showed that *GAssist* still can be improved in some aspects. Its comparison with *XCS* identified the class of datasets that it has difficulties in learning them. Nevertheless, this issue does not invalidate the contributions presented in this thesis because most of these contributions are not affected directly by this problem. Perhaps the only method that has some relation with this problem is the *ILAS* windowing scheme, and if we properly develop the stratification success model, this issue can be fixed.

Chapter 11

Further work

As the previous chapter did for the conclusions of the thesis, this one will to the same for the further work. First, the further work for each of the four kinds of contributions presented in this thesis will be reproduced. Next, global further work will be proposed.

11.1 Further work of the explicit default rule mechanism

From the studied policies for determining the class used in the default rule, the automatic policy was the best for its balance of accuracy and run-time. However, in order to achieve good performance, it had to increase the population size due to the niching mechanism it uses and the mating restriction introduced to avoid creating lethals. It would be useful to study if there is any feasible way to successfully recombine individuals with different default class. If we achieve this objective, maybe we can reduce the population size requirements of the *auto* policy.

Another alternative is developing more sophisticated heuristics to combine the simple policies, although they have more computational cost, because the tests show that it cannot choose correctly the suitable policy in all datasets. More interesting would be to develop a method that would only need some short runs, instead of running a full test for each candidate policy. Of course, it is clear that this approach would require a very solid statistical validation in order to assure that the decision taken is correct.

11.2 Further work on the ADI knowledge representation

The first item of further work proposed in chapter 6, checking the performance of the proposed groups of discretizers on a larger set of domains, was already made in the global comparison experiments reported in chapter 9. This global comparison did not find any significant differences between the groups of discretizers, as all of them had good performance. However, most of the new datasets of that comparison had a high number of nominal attributes and therefore, it is still worth doing a larger comparison specifically focused on the *ADI* groups.

The rest of further work deals with assuring that the dynamics of *ADI* assure that a proper learning process is achieved. This means assuring that the most suitable discretization algorithm for each dataset gets selected. This does not always happens because in some runs even the best discretizer can disappear from the population. The reinitialize operator was introduced to fix this problem, and some experimentation was made to tune its parameters. However, it is still worth analyzing other ways to tune this operator, or to find other less destructive operators than the reinitialize one to fix the problem of the dynamics of *ILAS*.

Instead of *fixing* this problem, maybe the alternative is to *avoid* it, by implementing some kind of niching mechanism, analogous to the one used for the default rule that is able to assure that all discretization algorithms survive until a fair competition can be applied. The problem is that performing a niching process over attributes is difficult, because an individual can in most datasets contain hundreds of attributes. If we use traditional recombination operators, this task is very difficult. Maybe, in order to deal with this question we should transform our system into a kind of estimation of distribution algorithm (Larranaga & Lozano, 2002).

11.3 Further work of the windowing methods

The main further work for the *ILAS* windowing scheme (or similar alternatives) is to find the ways to be able to use the information extracted from the stratification success model to tune the number of strata used on each dataset. Our hypothesis to achieve this objective would be to find a way to estimate the number of niches in the domain from the number of rules learned by the system on some short tests. This means filtering the rule set to discard over-specific rules and merge rules that cover subsets of the same niche.

Why is this objective so important? So far we have found experimentally the general settings of *ILAS* that maximize the performance of the system *in most datasets*, but not on all

them. If this model could be used on real datasets we could potentially extract the maximum performance from *ILAS*, creating an even more competent system.

11.4 Further work of the bloat control and generalization pressure methods

This further work, as the title of this subsection, is split in two parts. Small number of experiments were performed focused specifically on the rule pruning operator. We only illustrated how it increases the diversity of the population, and a 'rule of thumb' method was proposed to adjust its working parameters, based on the beneficial presence of a small quantity of neutral code in the individuals. This issue needs further study. Is the neutral code always beneficial? What quantity of dead rules can we leave in each individual? How does all this combine with the default rule mechanisms, that potentially introduce important changes to the distribution of examples covered by each rule. All these questions have yes to be answered.

Regarding the generalization pressure methods, the comparison made showed that the MDL-based fitness function is the best method, although we still have to wonder how general can this conclusion be. Does it only affect the exact settings used in these experiments or can it be generalized. Some more tests, changing the settings of the other generalization pressure methods in the ways suggested in chapter 8 should determine in general which is the best method.

Also, the *MDL* method can be improved in two ways. The first one affects the adaptive heuristic proposed to adjust the parameter that balances accuracy in complexity in this fitness function. This heuristic can potentially suffer from over-learning because it lacks an stop criteria to its function. The second way is to analyze the balance created by the current formulations of theory length, the complexity measure used by *MDL*. This formulation is crucial to promote the solutions to the classification process that we consider more suitable, and in some datasets this means balancing more than one factor, thus needing the proper equilibrium between the factors.

11.5 Global further work of GAssist

The most important piece of further work for *GAssist* is obviously the problem identified by the global comparative tests in chapter 9, about the difficulties of this system in learning low-frequency rules. In that chapter several ways to fix this problem were proposed. The first one is to import some kind of covering operator already used in the *Michigan* approach. Other alternatives come from the machine learning field, and deal with ways to assure that the system really learns for itself these small rules, by either modifying the fitness function to explicitly reward the individuals that learn them or applying some kind of super-sampling/sub-sampling to alter the proportions of examples of each class, and thus increasing the frequency of these rules. These two solutions have one problem: they only solve half of the task, because the recombination operators of the *GA* still have to generate the rules covering the minority rules. The covering operator, of some other kind of smart recombination operators would be the most suitable solution.

The most crucial issue of all these solutions is to find the proper accuracy-complexity balance. Right now *GAssist* generates solutions of very reduced size and this capacity should not be lost if it is not in exchange for an important performance boost.

Looking at all this description of further work, the same concept has appeared in several context as the way to improve *GAssist*: smart recombination. In the general context of evolutionary computation this is an issue that has been studied for more than ten years now, with names such as *estimation of distribution algorithms* (Larranaga & Lozano, 2002) or *competent genetic algorithms*, and generally (and this is probably a too simplistic definition) means using machine learning and statistics techniques to learn the structure of the problem being solved and allow the system to explore better the search space by creating smart exploration operators.

In GBML we are already in a machine learning context, but blind (or at least not completely smart) recombination operators are still used, especially in the *Pittsburgh* approach, because recently there has been some work (Butz, 2004) in integrating *competent genetic algorithms* into *XCS*. Thus, maybe it is time to take one step forward and introduce more machine learning into the learning process of *GAssist*.

Part IV

Appendix

Appendix A

Full results of the experimentation with the ADI knowledge representation

A.1 Results of the ADI tests with a single discretizer

Table A.1: Results of the ADI tests with a single discretizer for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	80.9±2.8	75.6±5.4	7.7±1.0	33.4±5.0
ChiMerge 0.50	85.3±0.7	78.8±4.2	9.8±1.8	37.9±4.7
Uniform frequency 10	85.9±0.8	78.2±4.2	10.6±1.9	40.2±5.8
Uniform frequency 15	85.7±0.7	78.4±4.3	10.6±1.9	38.3±4.8
Uniform frequency 20	85.6±0.7	78.8±4.4	10.3±1.8	39.4±4.4
Uniform frequency 25	85.4±0.8	78.4±4.1	10.2±1.8	38.3±4.2
Uniform frequency 5	85.8±0.6	78.7±3.8	10.9±2.1	40.3±5.1
ID3	85.8±0.7	78.7±4.4	10.7±2.2	36.8±5.2
Màntaras	83.8±1.1	78.9±4.2	8.2±1.2	37.0±5.3
Fayyad	82.1±1.2	76.8±4.0	7.8±1.2	34.9±5.7
Random	74.0±6.2	71.6±7.2	7.0±1.4	31.3±5.3
Uniform width 10	86.3±0.6	79.0±4.2	11.3±2.2	40.6±5.4
Uniform width 15	86.5±0.6	79.0±3.6	11.4±2.0	41.0±5.0
Uniform width 20	86.5±0.7	78.2±4.0	11.3±2.0	41.3±4.9
Uniform width 25	86.6±0.7	78.6±4.4	11.6±2.1	41.4±5.1
Uniform width 5	85.8±0.6	78.9±3.9	10.6±1.9	39.1±5.3
USD	75.9±0.3	69.0±3.0	6.8±0.4	26.5±2.5

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.2: Results of the ADI tests with a single discretizer for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	79.7±1.7	63.7±8.3	8.0±1.3	40.2±5.5
ChiMerge 0.50	83.6±1.9	64.8±7.2	9.4±1.8	44.7±6.0
Uniform frequency 10	83.1±1.7	65.5±8.3	10.1±1.7	43.4±5.4
Uniform frequency 15	83.5±1.4	63.5±8.2	10.0±2.0	44.0±5.9
Uniform frequency 20	84.1±1.8	64.8±7.4	10.0±1.9	43.9±5.7
Uniform frequency 25	83.8±1.7	65.2±7.0	10.1±1.8	44.9±6.2
Uniform frequency 5	79.1±1.1	63.7±7.0	9.0±1.5	41.1±4.7
ID3	84.9±1.6	65.3±7.4	9.6±1.8	42.3±6.7
Màntaras	58.6±1.5	57.8±1.3	2.0±0.0	10.3±2.0
Fayyad	63.4±2.1	59.6±4.8	3.4±1.3	24.2±7.1
Random	81.8±2.5	64.1±7.8	9.4±1.7	42.7±6.3
Uniform width 10	79.2±1.5	63.1±7.7	8.4±1.5	39.7±4.9
Uniform width 15	80.1±1.9	64.3±8.8	9.0±1.5	41.0±5.1
Uniform width 20	81.5±1.5	63.2±7.3	9.2±1.6	42.2±5.1
Uniform width 25	81.4±1.5	63.1±7.5	9.2±1.6	40.1±5.3
Uniform width 5	70.5±1.3	56.1±8.1	7.6±1.2	39.2±5.1
USD	83.7±1.5	65.2±8.0	9.3±1.8	41.7±5.5

Table A.3: Results of the ADI tests with a single discretizer for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	82.1±1.9	68.9±9.1	6.8±0.9	82.7±4.8
ChiMerge 0.50	83.1±1.7	69.2±9.4	7.2±1.1	85.3±5.2
Uniform frequency 10	80.9±1.9	69.7±9.0	7.4±1.2	85.1±5.0
Uniform frequency 15	81.1±1.8	68.2±9.4	7.2±1.2	86.1±5.2
Uniform frequency 20	81.2±1.9	68.7±9.1	7.3±1.2	86.9±5.0
Uniform frequency 25	81.7±2.0	68.6±9.4	7.4±1.2	86.8±7.1
Uniform frequency 5	78.3±2.0	68.3±9.6	7.1±1.1	82.7±4.4
ID3	82.4±1.9	69.6±9.1	7.3±1.1	85.0±5.2
Màntaras	69.5±2.4	65.4±10.1	6.3±0.5	72.7±4.2
Fayyad	77.3±2.0	67.1±9.0	6.6±0.7	80.6±4.5
Random	80.9±2.2	68.2±9.2	7.3±1.1	85.6±5.7
Uniform width 10	74.4±3.0	64.6±9.5	6.9±0.9	84.6±5.3
Uniform width 15	77.9±2.0	66.7±9.6	6.9±0.9	82.7±4.8
Uniform width 20	80.5±1.8	68.3±10.2	7.1±1.0	84.2±5.3
Uniform width 25	78.4±2.1	66.9±9.6	7.2±1.0	84.6±5.2
Uniform width 5	67.0±3.6	56.9±8.6	7.1±1.0	80.3±4.5
USD	82.5±2.0	68.5±9.3	7.4±1.1	83.9±5.3

Table A.4: Results of the ADI tests with a single discretizer for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	89.5±1.0	79.6±8.3	6.8±0.9	23.8±2.2
ChiMerge 0.50	90.2±1.0	78.2±7.7	7.1±1.1	24.0±2.3
Uniform frequency 10	91.7±1.2	78.9±7.3	7.7±1.2	24.3±2.3
Uniform frequency 15	91.4±1.3	78.2±7.7	7.6±1.0	24.3±2.7
Uniform frequency 20	91.0±1.4	78.2±7.0	7.6±1.1	24.2±2.7
Uniform frequency 25	90.9±1.3	78.1±7.1	7.8±1.2	24.4±2.5
Uniform frequency 5	91.6±1.0	80.2±7.1	7.4±1.1	24.8±2.2
ID3	91.5±1.0	79.8±7.2	7.3±1.2	24.7±2.0
Màntaras	88.7±0.9	83.4±6.7	6.4±0.7	21.9±2.2
Fayyad	88.8±0.9	82.6±7.2	6.4±0.6	23.9±3.4
Random	86.1±2.4	73.9±9.0	7.2±1.1	23.7±3.0
Uniform width 10	91.6±1.0	80.4±8.0	7.5±1.0	25.1±2.5
Uniform width 15	91.6±1.1	79.4±7.3	7.5±1.1	25.1±2.6
Uniform width 20	91.6±1.1	81.2±7.3	7.6±1.0	25.1±2.3
Uniform width 25	91.8±1.2	80.6±8.2	7.8±1.1	25.7±2.5
Uniform width 5	91.0±1.0	80.5±7.4	7.2±1.1	24.9±2.5
USD	90.3±1.1	79.4±7.6	7.0±0.9	23.7±2.2

Table A.5: Results of the ADI tests with a single discretizer for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	98.1±0.9	90.0±5.1	3.2±1.1	62.5±12.7
ChiMerge 0.50	98.3±0.7	89.8±5.2	3.2±1.1	62.7±11.7
Uniform frequency 10	95.9±1.4	90.6±4.8	4.2±1.4	60.2±10.8
Uniform frequency 15	96.2±1.1	89.7±5.4	4.2±1.2	62.6±11.6
Uniform frequency 20	96.5±1.1	89.5±5.2	4.2±1.3	62.1±11.0
Uniform frequency 25	96.7±0.9	89.9±5.2	4.3±1.3	63.8±9.4
Uniform frequency 5	94.7±1.1	88.0±5.3	4.5±1.2	61.0±7.7
ID3	97.7±0.9	90.1±5.5	3.7±1.1	62.2±11.5
Màntaras	94.4±1.4	90.8±4.5	3.3±1.1	44.2±11.0
Fayyad	97.3±0.5	91.5±5.0	2.4±0.6	54.9±10.0
Random	97.2±1.0	90.1±5.6	4.0±1.0	64.8±8.5
Uniform width 10	97.2±0.8	92.5±4.8	3.3±1.0	56.3±11.6
Uniform width 15	97.2±0.8	90.7±5.0	3.3±1.1	56.8±12.5
Uniform width 20	97.5±0.8	91.8±5.1	3.4±1.1	57.5±13.5
Uniform width 25	97.6±0.8	91.5±4.9	3.4±1.3	57.7±14.1
Uniform width 5	96.5±1.0	91.8±5.4	3.5±1.1	53.7±11.1
USD	97.7±0.9	90.5±5.2	3.7±1.3	61.1±11.0

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.6: Results of the ADI tests with a single discretizer for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	96.7±1.0	93.9±5.8	3.1±0.3	3.6±0.5
ChiMerge 0.50	97.7±0.8	94.8±5.7	3.3±0.6	4.0±0.5
Uniform frequency 10	97.4±0.7	95.2±5.9	3.6±0.7	4.0±0.6
Uniform frequency 15	98.0±0.7	94.7±5.9	3.5±0.7	4.3±0.7
Uniform frequency 20	97.9±0.9	94.9±5.9	3.6±0.7	4.3±0.7
Uniform frequency 25	96.6±0.9	93.8±5.8	3.3±0.5	3.9±0.6
Uniform frequency 5	96.3±1.0	93.1±6.4	3.7±0.8	4.1±0.5
ID3	98.2±0.8	95.0±5.8	3.6±0.7	3.8±0.5
Màntaras	97.3±0.9	95.7±5.5	3.4±0.6	3.9±0.6
Fayyad	96.8±0.9	93.8±5.9	3.2±0.4	4.1±0.6
Random	96.3±2.9	94.1±6.2	3.6±0.7	3.9±0.5
Uniform width 10	97.5±0.6	95.3±5.5	3.4±0.6	3.9±0.5
Uniform width 15	98.0±0.7	96.6±5.1	3.3±0.5	3.9±0.5
Uniform width 20	97.4±0.6	95.5±5.6	3.4±0.6	4.0±0.5
Uniform width 25	97.7±0.8	94.5±6.1	3.6±0.6	4.0±0.6
Uniform width 5	94.9±0.7	93.5±6.4	3.1±0.3	3.6±0.5
USD	97.7±0.8	94.3±5.7	3.3±0.5	3.6±0.5

Table A.7: Results of the ADI tests with a single discretizer for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	76.1±0.9	68.5±4.7	8.4±1.7	91.9±8.9
ChiMerge 0.50	76.2±0.9	68.8±5.0	8.8±1.7	93.4±9.0
Uniform frequency 10	75.9±0.9	68.9±4.7	9.2±1.9	90.5±8.6
Uniform frequency 15	76.2±0.9	68.8±4.6	9.1±1.7	92.3±8.0
Uniform frequency 20	76.7±0.9	68.8±5.2	9.5±1.9	93.0±7.5
Uniform frequency 25	76.5±0.9	68.8±4.8	9.0±1.8	92.2±8.1
Uniform frequency 5	73.8±0.9	67.5±4.1	8.4±1.6	88.9±7.5
ID3	76.9±1.0	69.2±5.2	9.2±1.9	88.1±8.2
Màntaras	74.1±0.7	68.0±4.3	8.5±1.6	90.2±9.5
Fayyad	76.5±0.8	69.4±5.4	7.7±1.1	91.1±9.3
Random	76.3±1.2	69.3±5.3	9.0±1.8	90.0±8.9
Uniform width 10	75.0±0.7	67.4±4.3	9.2±1.6	91.2±8.2
Uniform width 15	75.6±0.9	67.4±4.6	9.6±1.7	92.5±8.0
Uniform width 20	76.2±0.8	68.4±4.9	9.0±1.7	92.6±7.7
Uniform width 25	76.2±0.9	68.7±4.5	9.4±2.0	91.1±8.9
Uniform width 5	73.7±0.8	67.6±4.9	8.0±1.3	87.3±8.5
USD	76.9±0.9	69.0±4.9	9.3±1.9	87.2±8.2

Table A.8: Results of the ADI tests with a single discretizer for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	88.0±1.5	65.6±10.1	6.6±0.8	49.4±6.3
ChiMerge 0.50	88.7±1.5	65.4±10.1	6.9±1.1	50.9±6.4
Uniform frequency 10	84.8±1.5	65.9±10.8	7.0±1.0	50.3±5.6
Uniform frequency 15	85.5±1.4	66.9±10.1	7.0±1.0	49.4±5.7
Uniform frequency 20	85.9±1.5	67.3±11.3	7.1±1.1	50.1±6.1
Uniform frequency 25	86.1±1.4	66.6±10.7	7.0±0.9	50.2±6.0
Uniform frequency 5	83.0±1.4	68.0±9.9	6.9±1.0	47.3±5.4
ID3	87.6±1.5	64.9±10.8	7.1±1.1	50.6±6.6
Màntaras	73.8±2.9	66.5±10.3	6.3±0.6	41.4±3.5
Fayyad	74.3±1.8	65.3±11.1	6.2±0.5	43.4±3.5
Random	86.1±1.5	66.3±9.9	7.0±1.1	49.7±6.4
Uniform width 10	82.8±1.5	64.8±9.8	7.0±1.0	47.2±4.6
Uniform width 15	83.7±1.5	65.3±9.5	7.1±1.1	49.3±5.6
Uniform width 20	84.0±1.4	67.3±10.3	7.0±1.1	47.6±5.7
Uniform width 25	84.1±1.6	64.0±9.6	6.9±0.9	49.6±5.4
Uniform width 5	80.4±1.3	69.3±8.6	6.7±0.8	46.2±4.6
USD	87.3±1.5	66.5±8.9	6.9±1.0	49.5±6.1

Table A.9: Results of the ADI tests with a single discretizer for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	83.4±0.9	73.8±4.9	9.0±1.7	111.9±17.4
ChiMerge 0.50	84.1±0.8	74.2±5.2	10.1±1.9	115.5±18.2
Uniform frequency 10	83.5±0.8	74.0±4.9	10.5±2.1	107.6±13.9
Uniform frequency 15	83.6±0.8	73.7±4.4	10.3±2.1	107.1±11.9
Uniform frequency 20	83.8±0.9	73.7±4.6	10.2±2.2	108.6±16.0
Uniform frequency 25	84.0±0.8	73.6±5.1	10.4±2.2	111.6±14.7
Uniform frequency 5	82.2±0.7	74.4±4.4	9.5±1.8	111.3±14.8
ID3	84.2±0.8	73.7±4.8	9.5±1.9	104.6±15.5
Màntaras	79.1±0.9	74.9±5.1	6.0±1.0	91.4±8.6
Fayyad	80.5±0.7	73.4±4.9	6.1±0.9	103.3±13.6
Random	83.6±0.8	74.2±4.8	9.3±2.0	109.3±15.6
Uniform width 10	82.9±0.8	74.5±4.8	10.1±1.9	108.4±14.2
Uniform width 15	83.2±0.8	74.4±5.2	10.0±1.9	108.0±13.2
Uniform width 20	83.5±0.9	74.5±4.5	10.0±2.0	105.6±14.6
Uniform width 25	83.4±0.9	74.5±5.1	10.1±2.2	106.3±15.0
Uniform width 5	81.1±0.7	74.6±4.7	8.4±1.6	101.6±13.0
USD	84.0±0.7	74.3±5.0	9.3±1.9	97.1±15.2

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.10: Results of the ADI tests with a single discretizer for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	99.1±0.5	92.6±6.1	5.4±0.6	8.6±1.0
ChiMerge 0.50	99.4±0.5	92.4±5.0	5.4±0.6	8.8±1.0
Uniform frequency 10	98.8±0.6	92.2±5.9	5.5±0.7	8.8±1.0
Uniform frequency 15	99.0±0.7	92.9±5.1	5.6±0.7	8.8±1.0
Uniform frequency 20	98.9±0.6	93.2±5.1	5.6±0.7	8.8±1.0
Uniform frequency 25	99.1±0.6	92.8±5.0	5.4±0.6	8.8±1.0
Uniform frequency 5	97.8±0.8	93.5±5.5	5.3±0.5	8.8±1.0
ID3	99.1±0.7	92.9±5.3	5.4±0.6	8.5±0.9
Màntaras	98.3±0.8	91.8±5.7	5.4±0.6	8.8±0.9
Fayyad	98.2±0.5	91.9±5.4	5.3±0.5	8.8±1.0
Random	98.5±0.9	92.7±4.7	5.5±0.7	8.8±0.9
Uniform width 10	97.2±0.8	92.2±5.7	5.7±0.8	8.5±1.0
Uniform width 15	97.9±1.0	92.3±5.8	5.7±0.7	8.6±0.8
Uniform width 20	97.8±0.9	91.2±6.3	5.6±0.7	8.6±0.9
Uniform width 25	98.1±0.8	91.7±5.4	5.4±0.6	8.6±0.9
Uniform width 5	94.9±0.7	93.9±5.6	5.1±0.3	8.3±0.8
USD	98.9±0.7	93.4±4.4	5.5±0.6	8.3±0.9

Table A.11: Results of the ADI tests with a single discretizer for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	97.9±0.3	95.8±2.4	3.7±0.9	16.8±2.3
ChiMerge 0.50	98.6±0.3	95.8±2.2	3.6±0.9	17.7±2.1
Uniform frequency 10	98.5±0.3	95.6±2.3	3.7±0.8	17.3±2.2
Uniform frequency 15	98.3±0.3	95.8±2.5	3.8±0.9	16.9±2.6
Uniform frequency 20	98.1±0.4	95.8±2.4	3.3±0.8	16.2±2.5
Uniform frequency 25	98.0±0.5	95.6±2.5	2.9±0.7	15.4±2.5
Uniform frequency 5	98.2±0.3	95.7±2.4	3.5±0.6	17.1±1.9
ID3	98.5±0.3	95.7±2.4	3.5±0.7	15.7±2.1
Màntaras	98.2±0.3	95.6±2.5	4.3±0.9	17.0±2.0
Fayyad	97.9±0.3	95.9±2.2	3.5±0.9	17.1±2.2
Random	97.8±0.6	95.7±2.5	3.5±1.0	16.5±2.3
Uniform width 10	98.5±0.3	95.6±2.7	3.5±0.9	17.2±2.1
Uniform width 15	98.5±0.4	95.9±2.4	3.5±0.7	16.9±2.1
Uniform width 20	98.4±0.4	95.7±2.7	3.5±0.7	17.0±2.5
Uniform width 25	98.4±0.4	96.1±2.4	3.3±0.8	16.8±2.4
Uniform width 5	98.3±0.4	95.7±2.5	3.7±0.7	17.2±2.4
USD	97.9±0.3	95.9±2.1	3.2±0.5	14.3±1.8

Table A.12: Results of the ADI tests with a single discretizer for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	98.9±0.4	94.3±2.9	5.0±1.2	77.8±14.4
ChiMerge 0.50	98.8±0.4	94.4±2.9	4.9±1.0	77.5±15.6
Uniform frequency 10	98.6±0.5	94.7±2.8	5.8±1.5	69.6±15.6
Uniform frequency 15	98.6±0.5	94.6±2.5	5.5±1.2	71.9±14.2
Uniform frequency 20	98.7±0.4	94.3±3.1	5.6±1.3	71.1±12.9
Uniform frequency 25	98.7±0.5	94.0±3.0	5.5±1.3	72.4±13.5
Uniform frequency 5	98.2±0.5	93.9±3.5	5.6±1.4	63.6±13.1
ID3	98.9±0.4	94.2±3.0	5.2±1.2	69.9±15.3
Màntaras	98.2±0.5	94.5±3.1	4.8±0.9	56.0±10.1
Fayyad	98.2±0.4	94.6±2.8	4.4±0.8	55.8±9.6
Random	98.6±0.5	94.0±3.0	5.0±1.1	70.4±15.7
Uniform width 10	97.5±0.7	93.9±2.9	4.9±1.1	56.1±10.8
Uniform width 15	98.1±0.6	94.4±3.3	4.9±1.0	60.3±10.6
Uniform width 20	97.9±0.7	94.1±2.8	4.9±1.0	59.6±11.9
Uniform width 25	97.9±0.6	94.0±3.1	4.9±1.1	58.9±10.5
Uniform width 5	96.9±0.6	93.6±3.1	4.8±1.0	54.3±9.5
USD	98.8±0.5	94.2±3.2	5.0±1.2	67.5±15.8

Table A.13: Results of the ADI tests with a single discretizer for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	99.8±0.3	93.0±6.1	3.4±0.6	13.9±1.5
ChiMerge 0.50	99.8±0.3	93.7±5.6	3.7±0.7	15.0±1.6
Uniform frequency 10	99.4±0.4	93.8±5.4	4.1±0.8	14.9±1.7
Uniform frequency 15	99.5±0.5	93.1±5.9	4.0±0.8	15.1±1.7
Uniform frequency 20	99.5±0.4	92.9±6.4	3.9±0.8	15.1±1.8
Uniform frequency 25	99.6±0.4	93.7±5.7	4.0±0.8	15.3±1.5
Uniform frequency 5	99.1±0.5	93.6±5.3	4.2±0.6	14.7±1.4
ID3	99.7±0.4	93.7±5.5	3.9±0.8	15.0±1.7
Màntaras	99.2±0.6	93.7±5.7	3.9±0.7	14.0±1.4
Fayyad	99.8±0.4	93.4±5.8	3.4±0.7	13.3±1.2
Random	99.6±0.5	93.8±6.4	4.0±0.8	15.2±1.5
Uniform width 10	99.4±0.6	94.0±5.3	4.1±0.9	14.5±1.4
Uniform width 15	99.7±0.5	94.2±5.5	4.2±0.9	14.8±1.5
Uniform width 20	99.6±0.5	92.2±6.6	4.2±0.9	14.7±1.5
Uniform width 25	99.5±0.5	92.9±6.2	4.1±0.8	14.9±1.5
Uniform width 5	99.5±0.5	94.5±6.0	3.9±0.8	14.3±1.3
USD	99.6±0.4	93.2±5.6	4.1±0.7	14.9±1.5

Table A.14: Results of the ADI tests with a single discretizer for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
ChiMerge 0.01	94.0±1.8	68.3±9.5	4.2±2.0	40.3±9.0
ChiMerge 0.50	93.9±1.6	69.8±9.7	5.4±2.0	43.6±7.1
Uniform frequency 10	92.1±1.6	72.8±8.6	6.4±1.4	42.7±5.3
Uniform frequency 15	92.1±1.7	72.5±9.2	6.6±1.4	43.3±6.0
Uniform frequency 20	92.2±1.8	73.4±8.6	6.7±1.6	44.8±7.4
Uniform frequency 25	92.2±1.9	73.6±8.6	6.5±1.4	42.5±6.4
Uniform frequency 5	91.2±1.9	74.2±8.7	6.0±1.3	40.5±5.0
ID3	92.5±1.7	73.2±8.8	6.7±1.5	43.7±9.3
Màntaras	76.5±1.1	76.0±4.4	3.0±0.1	16.3±1.7
Fayyad	78.0±2.0	74.5±6.1	2.7±0.5	18.9±4.7
Random	91.4±1.7	71.9±8.5	6.4±1.3	41.8±7.5
Uniform width 10	89.8±2.1	73.0±8.1	5.9±1.4	36.8±7.9
Uniform width 15	89.6±2.5	72.8±9.1	6.0±1.4	37.1±8.4
Uniform width 20	89.1±2.5	73.5±8.6	5.8±1.6	36.1±9.2
Uniform width 25	89.6±2.2	74.0±9.6	6.1±1.4	37.1±8.0
Uniform width 5	89.7±1.8	73.6±8.9	5.4±1.1	35.3±5.1
USD	92.2±1.8	71.4±8.7	6.5±1.6	42.1±6.8

A.2 Results of the ADI tests with groups of discretizers without reinitialize

Table A.15: Results of the ADI tests with groups of discretizers without reinitialize for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	85.7±0.7	78.6±4.3	10.3±2.0	37.0±5.4
Group 2	85.3±0.8	79.3±3.9	9.6±1.8	36.5±5.2
Group 3	85.9±0.8	77.9±4.0	10.6±1.9	37.4±4.7
Group 4	86.0±0.7	78.4±4.1	11.0±2.0	37.7±5.3
Group 5	84.6±0.8	78.9±4.0	8.8±1.4	35.6±5.0
Group 6	86.0±0.7	78.4±4.4	10.3±1.8	38.3±5.1
Group 7	85.6±0.7	79.0±4.1	10.6±1.9	37.6±4.9

Table A.16: Results of the ADI tests with groups of discretizers without reinitialize for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	79.6±2.2	65.1±7.3	8.0±1.4	39.1±6.0
Group 2	76.3±3.0	63.7±7.3	7.3±1.3	38.0±7.4
Group 3	83.5±1.8	64.6±7.1	9.3±1.6	42.3±6.1
Group 4	83.1±1.8	65.4±7.0	9.1±1.6	42.1±6.8
Group 5	80.9±2.5	65.3±6.9	7.8±1.3	39.5±5.6
Group 6	79.1±2.2	62.7±8.5	8.4±1.5	39.3±5.6
Group 7	82.3±1.7	64.1±8.1	9.3±1.7	41.8±5.9

Table A.17: Results of the ADI tests with groups of discretizers without reinitialize for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	78.6±2.0	67.4±9.7	6.9±0.9	80.8±4.3
Group 2	77.0±2.2	66.1±9.3	6.7±0.7	79.0±5.0
Group 3	80.7±1.8	69.2±10.0	7.1±1.0	83.9±5.2
Group 4	80.4±1.9	67.8±8.7	7.0±1.0	84.6±5.9
Group 5	79.9±2.2	68.0±10.1	6.7±0.8	81.6±5.2
Group 6	76.7±2.0	66.8±9.9	6.9±1.0	80.9±5.0
Group 7	80.3±1.9	69.4±9.3	7.2±1.1	84.3±4.7

Table A.18: Results of the ADI tests with groups of discretizers without reinitialize for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	90.9±1.0	79.6±8.0	6.9±0.9	24.5±2.1
Group 2	90.3±1.0	80.5±7.1	6.9±1.0	23.6±2.4
Group 3	91.2±1.2	79.6±6.7	7.3±1.2	24.8±2.4
Group 4	91.5±1.1	80.0±7.5	7.4±1.2	24.9±2.5
Group 5	89.7±1.1	80.3±7.3	6.8±0.9	23.4±2.3
Group 6	91.4±1.1	80.6±7.5	7.5±1.1	25.1±2.7
Group 7	91.3±1.1	79.6±7.6	7.3±1.1	24.6±2.5

Table A.19: Results of the ADI tests with groups of discretizers without reinitialize for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	96.7±0.9	91.4±5.1	3.0±0.9	61.0±10.4
Group 2	96.7±0.9	91.1±4.7	2.9±0.9	57.2±11.0
Group 3	96.7±1.2	90.8±5.1	3.4±1.2	64.1±11.0
Group 4	97.0±1.0	91.7±4.7	3.3±1.2	61.0±10.7
Group 5	97.0±0.9	91.3±4.9	3.1±0.8	58.4±9.7
Group 6	96.4±1.1	90.6±5.1	3.2±1.0	61.9±11.5
Group 7	95.4±1.3	89.9±5.2	4.0±1.2	65.2±10.5

Table A.20: Results of the ADI tests with groups of discretizers without reinitialize for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.5±0.9	94.6±6.0	3.6±0.7	4.0±0.5
Group 2	97.5±1.1	94.2±5.9	3.6±0.6	4.0±0.5
Group 3	97.8±1.0	94.8±5.8	3.7±0.6	4.1±0.4
Group 4	97.8±0.9	94.3±5.7	3.7±0.6	4.1±0.5
Group 5	97.8±1.0	95.2±5.5	3.6±0.6	4.0±0.4
Group 6	97.6±0.8	95.0±5.9	3.6±0.6	4.1±0.5
Group 7	97.5±1.0	94.0±6.1	3.8±0.6	4.1±0.5

Table A.21: Results of the ADI tests with groups of discretizers without reinitialize for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	75.8±1.0	69.0±5.1	8.3±1.7	88.9±9.0
Group 2	75.9±0.8	69.7±5.1	7.9±1.5	87.6±8.0
Group 3	76.2±0.9	68.8±4.7	8.7±1.7	91.0±10.3
Group 4	76.2±0.8	69.0±4.9	8.4±1.6	89.3±8.3
Group 5	76.5±0.8	69.9±4.9	8.1±1.5	88.5±7.3
Group 6	75.4±0.9	68.1±5.0	8.2±1.4	88.1±8.3
Group 7	75.7±0.9	68.9±4.6	8.1±1.6	88.2±7.7

Table A.22: Results of the ADI tests with groups of discretizers without reinitialize for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.8±1.6	66.0±11.1	6.7±0.9	47.1±6.2
Group 2	82.8±1.6	67.5±10.5	6.6±0.9	44.9±5.0
Group 3	85.6±1.6	66.5±10.7	6.6±0.8	49.9±5.9
Group 4	85.5±1.5	65.5±9.8	6.9±1.1	49.4±5.6
Group 5	85.4±1.7	65.1±10.1	6.6±0.8	49.5±6.1
Group 6	83.1±1.7	66.6±10.5	6.7±0.8	47.8±5.7
Group 7	84.7±1.3	67.0±10.6	6.9±1.1	48.9±5.8

Table A.23: Results of the ADI tests with groups of discretizers without reinitialize for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	82.7±0.8	74.6±4.5	8.4±1.6	101.1±15.5
Group 2	82.0±0.9	74.4±4.8	7.5±1.3	98.1±11.8
Group 3	83.6±0.9	73.9±4.8	9.5±2.1	104.2±15.0
Group 4	83.6±0.8	74.0±4.8	9.7±1.9	103.6±16.2
Group 5	82.8±0.9	73.9±4.7	7.8±1.5	101.1±14.1
Group 6	82.5±0.8	74.3±4.7	9.2±1.8	104.7±14.8
Group 7	83.1±0.8	73.7±5.0	9.1±1.8	104.1±14.1

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.24: Results of the ADI tests with groups of discretizers without reinitialize for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.5±0.8	92.3±5.4	5.4±0.6	8.3±0.8
Group 2	98.5±0.6	92.4±5.9	5.3±0.5	8.1±0.9
Group 3	98.9±0.6	92.2±5.3	5.3±0.6	8.4±0.9
Group 4	98.8±0.7	92.4±5.2	5.3±0.5	8.3±0.9
Group 5	98.9±0.6	92.5±4.9	5.4±0.5	8.4±0.9
Group 6	97.5±0.8	91.6±5.8	5.4±0.7	8.2±0.8
Group 7	98.7±0.7	93.7±4.8	5.4±0.6	8.3±0.9

Table A.25: Results of the ADI tests with groups of discretizers without reinitialize for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.3±0.3	95.8±2.3	3.4±0.8	16.2±2.1
Group 2	98.2±0.3	95.9±2.3	3.5±0.7	15.6±2.1
Group 3	98.3±0.4	95.6±2.3	3.4±0.7	16.0±2.2
Group 4	98.4±0.4	95.6±2.3	3.5±0.7	16.0±2.0
Group 5	98.1±0.3	95.6±2.5	3.4±0.7	15.3±2.1
Group 6	98.4±0.3	95.9±2.5	3.2±0.5	16.0±2.0
Group 7	98.3±0.3	95.7±2.4	3.6±0.7	18.2±5.6

Table A.26: Results of the ADI tests with groups of discretizers without reinitialize for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.1±0.5	94.1±2.9	4.4±0.9	62.9±12.8
Group 2	98.1±0.5	94.6±2.9	4.5±1.0	57.2±10.6
Group 3	98.4±0.6	94.1±3.1	4.8±1.2	70.9±14.8
Group 4	98.5±0.6	94.2±3.1	4.9±1.4	65.0±12.9
Group 5	98.5±0.5	94.9±2.8	4.6±1.1	64.5±12.7
Group 6	97.6±0.7	94.2±3.1	4.4±0.9	60.0±11.4
Group 7	98.4±0.5	94.3±3.2	5.1±1.1	71.6±14.3

Table A.27: Results of the ADI tests with groups of discretizers without reinitialize for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.4±0.6	93.2±6.3	3.7±0.8	14.1±1.2
Group 2	99.5±0.5	93.4±6.1	3.6±0.8	13.7±1.2
Group 3	99.5±0.5	92.8±5.9	4.0±0.8	15.0±1.5
Group 4	99.5±0.5	93.2±5.7	4.0±0.7	14.9±1.6
Group 5	99.6±0.5	94.0±5.3	3.6±0.7	14.1±1.4
Group 6	99.1±0.6	93.3±5.5	4.1±0.8	14.5±1.1
Group 7	99.2±0.6	94.2±5.8	4.0±0.7	14.7±1.3

Table A.28: Results of the ADI tests with groups of discretizers without reinitialize for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	88.5±2.3	74.6±8.7	4.9±1.3	36.5±5.5
Group 2	86.6±2.4	75.2±9.2	4.3±1.2	33.1±5.4
Group 3	91.1±2.0	72.5±8.2	6.0±1.3	42.9±6.2
Group 4	90.6±1.8	73.1±9.1	5.9±1.4	41.5±6.6
Group 5	88.9±2.5	74.0±9.0	4.7±1.3	36.6±6.2
Group 6	89.2±2.0	72.8±9.9	5.3±1.3	37.3±5.4
Group 7	91.6±1.7	74.9±8.1	5.8±1.2	43.3±5.9

A.3 Results of the ADI tests with groups of discretizers with reinitialize

A.3.1 Reinitialize initial probability of 0.01

Table A.29: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	85.7±0.7	79.3±4.1	10.2±2.0	36.5±5.6
Group 2	85.3±0.8	79.0±4.1	9.7±1.7	36.0±5.4
Group 3	85.8±0.8	78.8±4.1	10.5±2.0	37.1±5.6
Group 4	85.8±0.7	78.7±4.2	10.4±1.9	36.1±5.2
Group 5	84.7±0.7	78.5±3.8	9.0±1.5	34.5±5.0
Group 6	85.9±0.7	79.1±4.2	10.4±1.9	37.7±5.5
Group 7	85.4±0.6	78.9±3.9	10.2±1.6	35.8±5.6

Table A.30: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	81.3±1.6	65.2±7.7	8.3±1.5	40.0±4.8
Group 2	79.0±1.9	65.0±6.1	7.5±1.3	39.2±5.9
Group 3	83.7±1.6	64.9±7.7	9.3±1.6	42.6±6.1
Group 4	83.5±1.7	65.1±7.4	9.2±1.6	41.6±6.2
Group 5	82.5±1.6	64.9±7.8	8.5±1.5	41.2±5.2
Group 6	80.1±1.7	61.8±8.4	8.3±1.5	39.2±5.3
Group 7	83.1±1.6	64.4±7.1	9.4±1.8	42.2±5.3

Table A.31: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	79.8±2.1	69.0±8.9	6.7±0.9	84.1±4.9
Group 2	78.6±2.1	66.5±10.4	6.7±0.8	83.6±6.1
Group 3	81.4±1.9	69.7±10.5	6.9±1.0	87.4±5.2
Group 4	81.5±1.8	69.5±10.0	6.9±0.9	87.2±5.4
Group 5	80.9±2.1	68.9±9.2	6.8±0.9	84.8±6.1
Group 6	78.3±1.9	67.3±10.1	6.7±0.8	84.5±4.9
Group 7	80.6±1.8	68.2±9.8	7.2±1.1	87.2±5.4

Table A.32: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	91.4±0.9	80.6±7.3	7.3±1.1	25.3±2.4
Group 2	90.9±0.9	81.2±6.9	7.1±1.0	24.5±2.3
Group 3	91.9±1.0	79.8±7.9	7.3±1.1	25.4±2.8
Group 4	92.0±1.0	79.9±7.2	7.5±1.2	25.6±2.2
Group 5	90.3±1.0	80.6±6.8	7.0±0.9	24.0±2.3
Group 6	91.8±1.0	80.9±7.4	7.4±1.1	25.7±2.4
Group 7	91.8±1.0	79.7±7.6	7.4±1.2	25.5±2.2

Table A.33: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.1±0.6	92.1±4.8	2.5±0.7	57.1±10.9
Group 2	97.1±0.6	92.1±5.0	2.3±0.5	54.7±9.4
Group 3	96.8±0.9	91.4±5.1	2.9±1.1	61.2±12.2
Group 4	97.2±0.8	92.3±4.8	2.7±0.8	59.1±11.6
Group 5	97.2±0.7	91.6±5.1	2.3±0.5	54.3±9.7
Group 6	96.5±0.8	92.3±5.0	2.5±0.8	55.5±9.7
Group 7	95.7±1.2	90.0±4.8	3.7±1.1	63.9±10.0

Table A.34: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.8±0.9	93.9±6.2	3.7±0.7	4.1±0.5
Group 2	97.7±1.0	94.1±5.9	3.6±0.7	4.0±0.5
Group 3	98.0±0.9	94.9±5.8	3.7±0.6	4.2±0.4
Group 4	98.0±1.0	94.6±5.7	3.6±0.7	4.1±0.5
Group 5	98.0±0.9	94.5±6.0	3.6±0.5	4.0±0.5
Group 6	97.9±0.8	95.2±5.9	3.6±0.5	4.1±0.5
Group 7	97.7±0.9	94.8±6.0	3.8±0.6	4.2±0.5

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.35: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	76.2±1.0	69.3±4.7	7.9±1.6	90.9±9.0
Group 2	76.2±0.8	69.2±5.2	7.7±1.4	90.3±7.5
Group 3	76.2±0.8	68.8±5.2	8.5±1.5	92.3±9.6
Group 4	76.3±0.9	68.5±5.3	8.6±1.5	90.8±8.6
Group 5	76.6±0.8	69.5±5.3	8.0±1.3	90.2±8.8
Group 6	75.7±0.9	69.1±5.0	8.2±1.6	91.4±8.9
Group 7	75.9±1.0	68.5±4.9	8.2±1.7	89.9±9.3

Table A.36: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	84.6±1.3	67.4±9.7	6.7±1.0	47.7±5.3
Group 2	83.5±1.4	68.0±9.6	6.4±0.8	45.1±4.9
Group 3	86.0±1.4	67.3±10.6	6.8±0.9	50.4±6.0
Group 4	85.9±1.5	66.8±9.9	6.7±0.8	49.5±5.7
Group 5	86.1±1.3	65.3±10.4	6.6±0.8	49.5±5.8
Group 6	84.1±1.4	68.5±10.2	6.6±0.8	49.1±5.2
Group 7	85.4±1.2	67.5±10.3	6.9±1.0	49.7±5.8

Table A.37: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.2±0.8	74.4±4.6	8.3±1.7	105.8±13.7
Group 2	82.8±0.7	74.6±4.5	7.4±1.4	101.5±11.7
Group 3	84.0±0.8	74.5±5.2	9.9±2.1	108.2±15.2
Group 4	83.9±0.8	74.6±4.7	9.6±2.0	105.3±13.6
Group 5	83.4±0.7	74.4±5.0	8.1±1.6	104.3±14.3
Group 6	83.0±0.8	74.3±4.5	9.1±1.9	106.9±15.0
Group 7	83.4±0.7	74.3±4.8	9.2±1.9	107.8±14.8

Table A.38: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.0±0.5	92.2±5.5	5.4±0.5	8.5±0.9
Group 2	98.9±0.5	92.1±5.6	5.4±0.6	8.4±0.9
Group 3	99.0±0.6	92.3±5.6	5.4±0.6	8.7±1.0
Group 4	99.1±0.6	92.4±5.8	5.4±0.6	8.6±0.9
Group 5	99.1±0.5	91.9±5.0	5.4±0.5	8.6±1.0
Group 6	98.0±0.7	92.1±6.0	5.3±0.5	8.4±0.9
Group 7	99.1±0.6	93.6±4.4	5.4±0.6	8.6±1.0

Table A.39: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.3±0.3	95.9±2.3	3.3±0.6	16.2±2.2
Group 2	98.2±0.4	95.9±2.2	3.4±0.7	15.6±2.1
Group 3	98.3±0.4	96.0±2.2	3.2±0.6	16.2±2.2
Group 4	98.4±0.3	95.6±2.3	3.3±0.6	16.1±1.9
Group 5	98.2±0.3	95.7±2.1	3.3±0.7	15.3±2.0
Group 6	98.3±0.4	95.8±2.3	3.2±0.6	16.1±2.2
Group 7	98.3±0.3	95.8±2.5	3.3±0.5	16.4±2.7

Table A.40: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.2±0.6	94.3±3.0	4.4±0.8	62.3±11.2
Group 2	98.2±0.5	94.2±3.1	4.2±0.8	56.2±10.0
Group 3	98.5±0.5	94.4±2.9	4.7±1.1	69.8±13.4
Group 4	98.4±0.5	94.2±2.9	4.6±1.1	63.3±13.0
Group 5	98.4±0.5	94.1±3.4	4.3±0.9	59.6±11.3
Group 6	97.5±0.5	94.0±3.1	4.2±0.8	62.0±11.4
Group 7	98.4±0.5	94.5±3.1	5.0±1.2	70.2±13.8

Table A.41: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.6±0.5	93.4±5.5	3.5±0.7	14.3±1.3
Group 2	99.8±0.4	93.4±5.4	3.4±0.6	14.1±1.2
Group 3	99.6±0.5	93.1±5.7	3.7±0.7	15.0±1.4
Group 4	99.6±0.5	93.8±5.4	3.8±0.7	15.0±1.4
Group 5	99.8±0.4	94.3±4.9	3.3±0.6	14.3±1.4
Group 6	99.3±0.6	93.1±6.0	3.9±0.7	14.9±1.5
Group 7	99.3±0.7	92.8±6.2	3.7±0.7	15.0±1.5

Table A.42: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.01 for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	89.4±1.8	74.1±8.3	4.7±1.1	38.0±5.7
Group 2	87.1±2.2	75.8±8.3	3.8±1.1	33.2±4.8
Group 3	91.7±1.7	73.1±8.7	5.6±1.3	42.8±6.6
Group 4	91.3±1.7	71.6±9.9	5.4±1.3	40.7±6.2
Group 5	90.0±2.1	73.8±8.6	4.3±1.2	36.6±5.4
Group 6	89.2±2.2	73.9±9.2	4.9±1.3	38.6±5.5
Group 7	92.0±1.8	73.9±8.3	5.3±1.3	42.3±5.2

A.3.2 Reinitialize initial probability of 0.02

Table A.43: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	85.5±0.8	78.4±3.9	9.7±1.8	35.6±5.3
Group 2	85.2±0.7	78.9±3.8	9.7±1.8	34.7±5.8
Group 3	85.7±0.8	78.7±4.3	10.5±1.8	36.3±5.5
Group 4	85.8±0.9	78.5±4.2	10.2±2.0	35.7±5.2
Group 5	84.7±0.8	79.5±3.8	9.1±1.5	33.8±5.1
Group 6	85.7±0.8	79.1±4.0	10.3±1.8	36.3±5.5
Group 7	85.3±0.7	78.3±4.1	9.7±1.8	35.5±5.2

Table A.44: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	81.4±1.4	64.7±7.6	8.4±1.4	40.1±4.8
Group 2	79.3±1.5	66.0±6.8	7.6±1.3	38.6±5.5
Group 3	83.8±1.6	64.7±7.3	9.6±1.9	41.6±5.6
Group 4	83.4±1.6	65.5±7.6	8.9±1.5	40.5±5.4
Group 5	82.6±1.6	64.1±6.5	8.4±1.5	40.4±4.3
Group 6	80.3±1.4	63.2±8.0	8.3±1.4	39.1±4.6
Group 7	82.6±1.6	64.7±7.8	9.1±1.6	40.8±5.7

Table A.45: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	79.8±1.9	69.0±8.9	6.7±0.8	84.8±5.1
Group 2	78.9±2.2	67.2±9.5	6.6±0.8	84.4±5.9
Group 3	81.2±2.0	68.9±9.9	6.9±1.0	87.7±5.2
Group 4	81.3±1.7	69.2±9.5	6.8±0.9	88.2±5.5
Group 5	80.6±2.2	67.5±9.9	6.7±0.9	85.7±6.2
Group 6	78.5±2.0	68.0±9.4	6.5±0.7	84.8±5.2
Group 7	80.5±2.0	69.1±9.6	6.8±0.9	88.0±5.4

Table A.46: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	91.5±1.0	80.2±7.1	7.2±1.0	25.8±2.4
Group 2	90.9±0.9	80.5±7.0	7.1±1.1	25.0±2.5
Group 3	92.1±0.9	80.6±7.0	7.4±1.1	25.8±2.2
Group 4	92.0±0.9	80.8±7.4	7.3±1.2	26.1±2.3
Group 5	90.1±0.9	80.8±7.3	6.9±1.0	24.5±2.0
Group 6	91.7±1.0	81.0±8.1	7.2±1.0	25.9±2.3
Group 7	91.8±1.0	80.0±7.5	7.5±1.2	25.7±2.1

Table A.47: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.0±0.7	92.1±4.9	2.2±0.5	56.8±9.5
Group 2	97.3±0.6	91.9±4.9	2.3±0.6	55.3±9.3
Group 3	96.6±0.7	92.7±4.5	2.5±0.9	57.4±10.0
Group 4	97.0±0.8	93.0±4.5	2.6±1.1	56.4±9.6
Group 5	97.3±0.7	91.4±5.0	2.2±0.4	53.5±8.2
Group 6	96.5±0.7	92.5±5.3	2.4±0.7	54.8±8.1
Group 7	95.0±1.2	90.6±5.1	3.5±1.1	59.1±9.3

Table A.48: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.8±0.9	94.4±6.0	3.6±0.7	4.2±0.5
Group 2	97.8±0.9	94.4±6.0	3.6±0.6	4.1±0.6
Group 3	98.1±0.9	94.3±6.1	3.8±0.6	4.2±0.5
Group 4	98.2±0.9	94.2±6.0	3.8±0.7	4.2±0.5
Group 5	98.1±0.8	94.4±6.2	3.6±0.6	4.1±0.5
Group 6	98.1±0.9	95.1±5.4	3.7±0.6	4.2±0.5
Group 7	97.8±1.0	94.0±6.6	3.8±0.6	4.3±0.5

Table A.49: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	76.2±0.8	69.4±4.6	8.0±1.5	92.0±9.2
Group 2	76.2±0.7	69.3±5.1	7.7±1.4	90.4±8.3
Group 3	76.1±0.9	68.9±5.1	8.5±1.7	92.8±9.4
Group 4	76.2±0.9	68.5±4.9	8.7±1.7	92.3±8.9
Group 5	76.5±0.9	69.7±4.7	7.9±1.6	92.6±8.0
Group 6	75.8±1.0	68.7±5.0	8.0±1.6	91.6±9.7
Group 7	75.6±0.8	69.2±4.7	8.3±1.6	90.8±8.8

Table A.50: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	84.4±1.3	66.9±11.5	6.6±0.9	48.3±5.1
Group 2	83.4±1.3	68.0±10.3	6.4±0.6	45.6±5.1
Group 3	85.9±1.2	66.4±11.2	6.7±0.9	50.8±5.9
Group 4	85.8±1.4	68.1±9.9	6.8±1.0	49.2±5.6
Group 5	85.7±1.3	65.5±10.8	6.7±0.9	49.5±6.2
Group 6	84.0±1.1	69.2±10.8	6.5±0.7	47.6±4.9
Group 7	85.1±1.3	67.8±10.9	6.9±1.1	50.2±5.1

Table A.51: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.5±0.7	74.6±5.2	8.5±1.9	106.2±14.1
Group 2	82.9±0.9	73.9±5.2	7.6±1.4	103.5±11.4
Group 3	84.0±0.8	73.9±5.6	9.3±2.0	109.0±13.9
Group 4	84.0±0.8	74.7±4.8	9.0±2.0	105.6±13.6
Group 5	83.4±0.7	74.4±5.0	8.1±1.7	104.2±13.1
Group 6	83.2±0.7	74.2±4.9	8.6±1.7	107.0±14.2
Group 7	83.5±0.8	74.5±4.8	8.9±1.9	105.5±13.6

Table A.52: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.9±0.6	92.1±5.7	5.3±0.5	8.6±0.9
Group 2	98.9±0.6	91.4±5.5	5.3±0.6	8.6±1.0
Group 3	99.3±0.6	92.7±5.5	5.4±0.6	8.8±0.9
Group 4	99.1±0.6	92.5±4.7	5.3±0.5	8.7±1.0
Group 5	99.2±0.5	92.8±4.6	5.4±0.6	8.8±0.9
Group 6	98.2±0.7	92.3±5.6	5.4±0.6	8.4±0.9
Group 7	99.0±0.6	92.9±5.0	5.4±0.6	8.8±0.9

Table A.53: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.2±0.4	95.9±2.4	3.2±0.6	15.7±2.3
Group 2	98.1±0.4	95.9±2.4	3.2±0.6	15.5±1.9
Group 3	98.3±0.4	95.6±2.4	3.1±0.5	15.9±2.1
Group 4	98.2±0.4	95.7±2.4	3.0±0.6	15.5±1.9
Group 5	98.1±0.4	95.8±2.2	3.1±0.7	15.0±1.9
Group 6	98.3±0.4	95.8±2.4	3.0±0.6	16.1±2.1
Group 7	98.2±0.4	95.6±2.5	3.1±0.6	16.3±2.7

Table A.54: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.1±0.5	94.2±3.0	4.1±0.8	63.6±11.7
Group 2	98.0±0.5	94.3±3.1	4.1±0.9	57.0±10.3
Group 3	98.3±0.6	94.0±3.0	4.3±1.0	69.7±13.6
Group 4	98.3±0.5	94.1±2.9	4.3±0.9	65.1±11.7
Group 5	98.4±0.5	94.5±3.0	4.2±0.9	63.5±11.6
Group 6	97.6±0.6	94.0±2.9	4.2±0.9	60.3±11.7
Group 7	98.3±0.5	94.3±3.1	4.8±0.9	70.9±14.6

Table A.55: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.7±0.4	92.8±5.6	3.4±0.7	14.6±1.3
Group 2	99.8±0.4	93.1±5.7	3.3±0.6	14.2±1.4
Group 3	99.7±0.4	93.4±6.1	3.6±0.6	15.2±1.3
Group 4	99.6±0.4	93.7±5.5	3.7±0.7	15.0±1.4
Group 5	99.8±0.3	93.5±5.6	3.2±0.4	14.6±1.3
Group 6	99.4±0.5	93.0±5.3	3.9±0.7	14.8±1.4
Group 7	99.3±0.5	93.3±6.1	3.6±0.7	15.1±1.4

Table A.56: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.02 for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	89.3±2.0	75.1±8.9	4.0±1.2	36.7±4.0
Group 2	87.3±2.0	75.9±8.4	3.5±1.1	33.0±3.9
Group 3	91.2±2.0	74.2±8.3	5.0±1.3	41.0±5.2
Group 4	91.1±1.8	73.8±7.5	4.9±1.2	38.9±4.7
Group 5	90.0±1.9	74.7±9.0	3.9±1.0	36.4±4.7
Group 6	89.6±2.0	72.9±8.2	4.8±1.2	37.9±4.5
Group 7	91.6±1.7	75.7±8.7	4.9±1.5	40.6±4.5

A.3.3 Reinitialize initial probability of 0.03

Table A.57: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	85.4±0.8	79.1±4.2	9.9±1.8	34.4±5.3
Group 2	85.1±0.7	79.3±3.5	9.7±1.9	34.2±5.5
Group 3	85.6±0.9	79.0±4.0	10.3±1.8	34.1±4.9
Group 4	85.6±0.8	79.3±4.3	10.2±1.7	34.9±5.1
Group 5	84.6±0.7	79.2±4.1	9.1±1.4	33.1±5.0
Group 6	85.6±0.8	78.7±4.2	10.2±2.0	34.4±5.4
Group 7	85.2±0.7	78.6±4.2	10.1±2.0	33.2±5.1

Table A.58: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	81.3±1.3	65.2±7.8	8.2±1.4	39.3±5.0
Group 2	79.3±1.5	66.8±6.9	7.5±1.3	38.3±5.6
Group 3	83.3±1.7	63.9±6.8	9.1±1.8	41.1±5.6
Group 4	83.2±1.5	64.7±7.6	8.9±1.6	40.8±5.0
Group 5	82.7±1.5	64.9±7.3	8.4±1.6	40.1±4.4
Group 6	80.3±1.5	62.8±7.5	8.4±1.5	38.8±4.7
Group 7	82.7±1.6	66.1±7.4	8.9±1.5	41.1±4.7

Table A.59: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	79.7±2.0	68.9±8.6	6.6±0.7	84.8±5.2
Group 2	79.3±2.3	67.3±9.8	6.7±0.8	84.3±6.0
Group 3	81.2±1.7	69.7±9.4	6.9±1.0	87.2±5.8
Group 4	81.0±2.0	69.5±10.0	6.9±1.0	87.2±6.0
Group 5	80.2±2.0	67.9±9.2	6.6±0.8	85.6±5.9
Group 6	78.5±2.0	68.4±9.4	6.8±0.8	85.0±5.7
Group 7	80.5±2.0	69.4±9.6	6.9±1.0	88.6±5.1

Table A.60: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	91.4±1.0	81.0±7.5	7.3±1.1	25.3±2.2
Group 2	90.7±1.0	81.1±7.7	7.0±0.9	24.4±2.5
Group 3	91.9±1.0	79.8±8.1	7.5±1.2	25.7±2.6
Group 4	91.6±1.1	80.5±6.8	7.2±1.1	25.9±2.1
Group 5	89.9±1.0	80.3±7.4	6.9±1.0	24.1±2.2
Group 6	91.6±0.8	80.2±7.1	7.4±1.2	25.8±2.4
Group 7	91.6±0.9	80.6±7.8	7.4±1.2	25.7±2.2

Table A.61: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.1±0.6	91.8±5.0	2.2±0.5	56.5±7.9
Group 2	97.2±0.5	92.3±5.0	2.1±0.3	54.5±8.3
Group 3	96.4±0.9	91.5±5.1	2.6±0.9	56.6±8.6
Group 4	96.8±0.7	92.5±4.5	2.4±0.9	56.5±7.9
Group 5	97.4±0.5	91.8±5.2	2.2±0.4	53.3±8.1
Group 6	96.5±0.7	92.7±5.1	2.3±0.7	55.1±8.1
Group 7	94.5±1.0	90.3±5.3	3.3±1.2	56.6±7.0

Table A.62: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.0±0.8	93.8±6.4	3.7±0.7	4.1±0.5
Group 2	97.9±0.9	94.4±5.9	3.5±0.6	4.1±0.5
Group 3	98.1±0.9	94.4±5.9	3.8±0.6	4.2±0.5
Group 4	98.2±0.9	94.1±5.9	3.7±0.6	4.2±0.5
Group 5	98.1±0.9	94.8±6.1	3.5±0.6	4.0±0.5
Group 6	98.0±0.9	95.3±5.5	3.6±0.6	4.1±0.5
Group 7	98.1±0.9	94.6±5.8	3.8±0.7	4.2±0.5

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.63: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	76.1±0.9	69.4±5.0	8.1±1.4	93.0±8.1
Group 2	76.0±0.8	69.9±4.7	7.6±1.2	90.8±8.2
Group 3	76.0±0.9	68.5±4.7	8.3±1.6	93.8±8.9
Group 4	76.0±0.9	68.3±5.0	8.5±1.5	92.0±8.2
Group 5	76.4±0.9	69.5±5.1	7.9±1.4	91.6±8.6
Group 6	75.7±0.9	69.1±5.0	8.0±1.4	92.2±9.9
Group 7	75.4±0.8	68.9±4.8	7.9±1.4	91.8±8.3

Table A.64: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	84.1±1.3	67.8±10.3	6.5±0.8	48.4±4.7
Group 2	83.4±1.3	68.7±11.2	6.5±0.7	46.0±4.4
Group 3	85.5±1.3	68.0±10.7	6.8±1.0	50.7±4.7
Group 4	85.5±1.4	67.9±9.4	6.8±1.0	49.1±5.1
Group 5	85.5±1.5	66.7±9.9	6.5±0.8	48.5±5.0
Group 6	83.8±1.3	70.3±9.5	6.7±1.1	48.3±3.9
Group 7	84.5±1.3	69.1±10.6	6.9±1.1	49.8±4.7

Table A.65: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.4±0.7	74.5±5.0	7.9±1.6	104.8±12.6
Group 2	82.9±0.8	74.9±4.9	7.6±1.4	103.1±11.6
Group 3	83.8±0.9	73.9±5.0	9.3±2.0	105.9±13.9
Group 4	83.9±0.8	73.8±5.1	9.3±1.8	103.3±13.7
Group 5	83.4±0.7	74.4±5.2	7.9±1.6	103.4±11.9
Group 6	83.2±0.8	74.4±4.7	8.9±1.9	104.6±13.4
Group 7	83.5±0.8	74.0±4.8	8.8±1.9	106.0±13.4

Table A.66: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.0±0.5	92.5±5.2	5.4±0.6	8.6±0.9
Group 2	99.0±0.5	92.1±5.4	5.3±0.5	8.5±0.9
Group 3	99.2±0.6	92.6±5.4	5.5±0.6	8.7±1.0
Group 4	99.1±0.6	92.3±5.3	5.3±0.6	8.5±0.9
Group 5	99.3±0.5	92.6±5.1	5.4±0.7	8.7±0.9
Group 6	98.2±0.7	91.3±5.6	5.4±0.6	8.4±0.8
Group 7	99.2±0.6	93.2±4.9	5.3±0.5	8.7±0.9

Table A.67: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.1±0.4	95.9±2.2	3.0±0.6	15.4±2.1
Group 2	97.9±0.4	96.0±2.3	3.0±0.7	14.7±1.8
Group 3	98.2±0.4	95.9±2.3	3.1±0.7	15.5±2.1
Group 4	98.2±0.4	96.0±2.3	3.1±0.7	15.1±1.8
Group 5	97.9±0.4	96.0±2.3	2.9±0.7	14.5±1.9
Group 6	98.2±0.4	95.9±2.4	3.0±0.7	15.7±2.3
Group 7	98.1±0.4	95.7±2.4	3.1±0.6	16.0±3.0

Table A.68: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.0±0.5	94.2±3.2	4.1±0.9	62.4±11.3
Group 2	98.0±0.5	94.0±3.1	3.9±0.7	54.9±9.3
Group 3	98.1±0.6	94.1±3.3	4.3±1.0	67.3±13.6
Group 4	98.2±0.6	94.2±3.2	4.3±1.1	62.6±11.7
Group 5	98.2±0.6	94.0±3.0	3.9±0.8	57.2±11.2
Group 6	97.5±0.7	94.0±2.9	3.9±0.8	59.5±12.5
Group 7	98.1±0.6	94.0±3.2	4.6±1.1	71.5±12.8

Table A.69: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.8±0.4	93.9±5.4	3.4±0.6	14.6±1.2
Group 2	99.9±0.3	93.8±5.6	3.2±0.5	14.3±1.3
Group 3	99.6±0.5	93.3±5.9	3.6±0.7	15.2±1.4
Group 4	99.6±0.4	93.9±5.4	3.5±0.6	14.9±1.5
Group 5	99.8±0.3	93.9±5.7	3.2±0.5	14.7±1.4
Group 6	99.3±0.5	93.6±5.9	3.5±0.6	14.7±1.3
Group 7	99.3±0.6	92.4±6.1	3.5±0.7	15.2±1.3

Table A.70: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.03 for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	89.0±2.1	74.9±8.5	3.8±1.1	35.3±4.2
Group 2	87.2±2.1	76.3±8.1	3.3±1.1	32.4±4.2
Group 3	91.5±1.8	73.8±8.8	4.8±1.2	39.4±5.0
Group 4	90.7±1.8	75.8±8.8	4.7±1.2	37.9±4.2
Group 5	90.1±2.0	74.1±8.6	3.8±1.2	35.7±4.5
Group 6	89.3±1.8	74.7±8.0	4.4±1.1	36.1±4.1
Group 7	91.1±1.7	74.9±8.9	4.4±1.1	39.0±3.9

A.3.4 Reinitialize initial probability of 0.04

Table A.71: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *bal* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	85.3±0.8	78.9±3.8	9.7±1.7	33.0±4.8
Group 2	84.8±0.8	79.0±3.8	9.5±1.7	32.0±4.6
Group 3	85.4±0.9	79.0±3.9	10.1±1.9	33.9±5.3
Group 4	85.5±0.7	79.1±4.3	10.0±1.8	33.5±4.5
Group 5	84.4±0.8	79.1±3.4	9.1±1.5	31.6±4.5
Group 6	85.4±0.7	78.9±3.9	9.8±1.7	33.2±4.6
Group 7	84.9±0.8	78.0±3.9	9.5±1.6	32.4±4.8

Table A.72: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *bpa* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	81.1±1.4	66.0±6.8	8.0±1.4	39.8±3.8
Group 2	79.2±1.3	65.7±6.6	7.5±1.4	37.6±5.2
Group 3	83.2±1.4	64.8±7.2	9.1±1.7	41.1±4.6
Group 4	82.8±1.5	66.1±7.5	9.0±1.5	40.0±4.8
Group 5	82.2±1.4	65.5±5.9	8.3±1.7	39.3±4.3
Group 6	79.7±1.6	62.4±8.0	8.3±1.5	37.9±3.7
Group 7	82.3±1.5	64.6±6.9	8.9±1.5	40.5±4.4

Table A.73: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *gls* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	79.9±2.1	68.1±9.8	6.8±0.8	85.8±5.1
Group 2	79.1±2.4	66.1±9.1	6.7±0.8	84.4±6.0
Group 3	81.1±2.1	68.8±9.7	7.1±1.0	88.0±6.0
Group 4	80.6±1.8	68.8±9.1	6.9±1.0	87.9±5.8
Group 5	80.3±1.9	67.5±9.3	6.6±0.7	85.6±6.0
Group 6	78.4±2.0	66.5±9.4	6.9±0.9	85.8±4.9
Group 7	80.2±2.1	68.8±10.2	7.0±1.0	88.8±5.9

Table A.74: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *h-s* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	91.1±0.9	80.8±7.2	7.4±1.2	25.5±2.4
Group 2	90.4±1.0	80.6±7.9	6.9±0.9	24.8±2.1
Group 3	91.6±1.0	81.3±7.9	7.4±1.2	25.9±2.2
Group 4	91.6±0.9	80.6±6.9	7.4±1.1	25.9±2.0
Group 5	89.7±1.0	80.3±7.5	6.8±1.1	23.9±2.0
Group 6	91.3±1.0	81.5±7.2	7.3±1.2	26.0±2.4
Group 7	91.4±0.9	81.2±7.5	7.5±1.3	25.8±2.3

Table A.75: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *ion* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.0±0.6	92.2±4.8	2.2±0.5	58.6±8.3
Group 2	97.1±0.5	92.2±4.5	2.1±0.3	55.3±7.9
Group 3	96.5±0.9	91.3±5.3	2.4±0.8	58.9±8.1
Group 4	96.8±0.8	92.5±4.8	2.3±0.6	57.7±8.3
Group 5	97.3±0.7	91.5±4.8	2.2±0.5	55.0±7.4
Group 6	96.4±0.7	92.5±5.1	2.3±0.7	57.5±7.1
Group 7	94.3±1.0	90.0±5.7	3.6±1.6	59.1±6.7

Table A.76: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *irs* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.0±0.9	94.7±5.9	3.6±0.6	4.1±0.5
Group 2	97.8±0.9	94.1±6.2	3.5±0.6	4.1±0.5
Group 3	98.2±0.8	94.8±6.2	3.7±0.6	4.1±0.5
Group 4	98.2±0.9	94.2±5.8	3.8±0.7	4.2±0.5
Group 5	98.1±0.9	94.8±5.8	3.6±0.6	4.1±0.5
Group 6	98.1±0.9	95.4±5.4	3.6±0.6	4.1±0.5
Group 7	97.8±1.0	94.6±6.2	3.7±0.7	4.2±0.5

Table A.77: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *lrn* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	75.9±0.9	69.5±4.7	7.9±1.4	92.5±8.9
Group 2	76.0±0.8	69.7±5.1	7.9±1.4	90.8±9.3
Group 3	76.0±0.9	68.8±5.0	8.3±1.4	93.3±9.0
Group 4	76.0±0.8	68.5±4.9	8.4±1.6	92.0±8.8
Group 5	76.4±0.8	69.5±4.9	7.9±1.6	91.4±8.7
Group 6	75.5±0.9	68.8±5.1	7.8±1.4	92.0±9.5
Group 7	75.4±0.8	68.9±4.9	8.0±1.6	91.7±9.3

Table A.78: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *mmg* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.9±1.4	67.8±9.7	6.5±0.7	48.3±4.6
Group 2	82.9±1.4	67.5±11.1	6.4±0.7	45.7±4.0
Group 3	85.2±1.3	67.7±11.3	6.8±0.9	49.8±5.3
Group 4	85.3±1.4	67.1±10.5	6.9±1.0	48.9±5.0
Group 5	84.9±1.4	66.8±9.9	6.5±0.8	48.6±4.8
Group 6	83.6±1.2	70.4±9.7	6.5±0.8	48.0±4.0
Group 7	84.5±1.2	67.8±11.3	7.0±1.1	49.6±4.2

Table A.79: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *pim* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	83.3±0.8	74.4±5.5	8.2±1.6	105.2±13.1
Group 2	82.8±0.8	74.3±4.6	7.5±1.3	102.8±10.7
Group 3	83.8±0.8	74.5±5.1	9.1±1.9	106.8±12.2
Group 4	83.7±0.7	74.5±4.9	9.2±1.7	104.3±11.6
Group 5	83.4±0.7	74.5±5.0	7.9±1.6	104.0±11.7
Group 6	83.0±0.7	74.4±4.4	8.4±1.7	105.1±12.9
Group 7	83.4±0.8	74.2±4.4	9.1±2.0	106.5±13.9

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Table A.80: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *thy* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.0±0.5	92.6±4.7	5.2±0.4	8.5±0.9
Group 2	99.0±0.5	92.0±5.4	5.4±0.6	8.6±0.8
Group 3	99.1±0.6	92.5±5.7	5.3±0.6	8.7±0.9
Group 4	99.2±0.6	92.0±5.0	5.3±0.5	8.5±1.0
Group 5	99.3±0.5	92.1±5.8	5.5±0.7	8.7±1.0
Group 6	98.2±0.7	91.9±5.2	5.4±0.6	8.5±0.8
Group 7	99.2±0.5	93.0±4.8	5.4±0.6	8.7±1.0

Table A.81: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *wbcd* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	98.0±0.4	96.0±2.4	3.0±0.6	15.2±2.0
Group 2	97.8±0.4	95.9±2.5	3.0±0.7	14.8±2.0
Group 3	98.1±0.4	96.1±2.5	3.0±0.6	15.3±2.2
Group 4	98.1±0.4	96.0±2.5	2.9±0.7	14.7±1.9
Group 5	97.8±0.4	95.9±2.2	2.8±0.6	14.5±1.6
Group 6	98.0±0.4	96.1±2.4	2.7±0.7	15.6±2.0
Group 7	97.9±0.4	95.8±2.4	3.0±0.7	15.7±2.8

Table A.82: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *wdbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	97.9±0.6	94.3±2.9	4.0±0.8	59.3±11.7
Group 2	97.9±0.5	94.0±2.9	3.9±0.8	55.3±9.2
Group 3	98.1±0.5	94.1±3.0	4.3±1.0	67.7±12.9
Group 4	98.1±0.6	94.2±2.9	4.2±1.0	60.2±11.0
Group 5	98.1±0.5	93.9±3.0	3.8±0.8	56.8±10.0
Group 6	97.3±0.6	93.9±2.8	3.9±0.8	60.6±10.6
Group 7	98.0±0.7	94.0±3.2	4.6±1.1	68.8±13.4

Table A.83: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *wine* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	99.7±0.4	93.8±5.4	3.3±0.6	14.8±1.3
Group 2	99.8±0.4	93.6±5.5	3.2±0.5	14.5±1.2
Group 3	99.6±0.5	93.4±5.7	3.4±0.6	15.2±1.4
Group 4	99.5±0.5	93.8±6.2	3.5±0.6	15.2±1.3
Group 5	99.8±0.3	93.8±5.7	3.2±0.4	14.8±1.5
Group 6	99.3±0.5	93.6±4.8	3.6±0.7	15.0±1.3
Group 7	99.3±0.6	92.9±5.9	3.5±0.7	15.5±1.4

Table A.84: Results of the ADI tests with groups of discretizers with reinitialize and prob. 0.04 for the *wpbc* dataset

Discretizer	Training acc.	Test acc.	#rules	Run-time (s)
Group 1	88.6±1.9	75.3±8.7	3.7±1.1	35.5±3.9
Group 2	87.1±2.1	76.3±8.6	3.0±1.0	32.0±3.7
Group 3	91.1±1.9	75.2±8.6	4.6±1.1	37.9±4.6
Group 4	90.6±1.9	73.5±9.1	4.6±1.2	37.2±4.2
Group 5	89.4±1.9	75.8±8.2	3.5±1.1	34.0±3.7
Group 6	88.9±1.8	75.2±8.7	4.2±1.1	35.7±4.0
Group 7	91.1±1.6	76.0±9.4	4.3±1.1	38.5±3.4

APPENDIX A. FULL RESULTS OF THE EXPERIMENTATION WITH THE ADI KNOWLEDGE REPRESENTATION

Appendix B

Full results of the experimentation with the ILAS windowing system

B.1 Results of the tests with the constant learning steps strategy

Table B.1: Results of the constant learning steps strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
bal	1	87.09±0.54	79.17±4.00	11.69±1.73	57.38±5.95
	2	86.49±0.60	78.89±4.07	11.52±2.24	61.77±7.67
	3	85.79±0.72	78.52±3.93	9.50±1.48	65.54±8.44
	4	85.38±0.66	79.29±3.88	8.63±1.33	69.97±8.28
	5	84.95±0.80	78.70±4.04	8.18±1.28	73.94±8.57
bpa	1	82.08±1.58	62.50±8.60	8.96±1.45	58.66±6.47
	2	82.50±1.44	63.70±8.11	8.60±1.45	65.84±6.33
	3	81.96±1.51	63.07±7.08	7.92±1.50	74.34±6.26
	4	81.13±1.56	61.23±7.08	7.17±1.07	81.88±5.80
	5	80.39±1.54	63.26±6.83	6.97±1.01	91.24±5.31
bre	1	89.41±1.75	70.80±8.53	12.04±1.62	43.06±7.37
	2	88.06±1.84	70.29±7.19	10.86±1.41	47.28±8.51
	3	86.36±2.14	70.65±7.48	9.56±1.33	50.67±9.77
	4	85.02±2.25	70.60±6.99	8.86±1.19	57.33±8.93
	5	86.05±2.07	71.24±7.46	8.95±1.53	60.88±8.72

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.1: Results of the constant learning steps strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
cmc	1	59.11±0.88	54.81±4.19	6.25±0.89	123.66±6.11
	2	59.66±1.10	54.77±4.06	10.71±3.83	130.89±9.53
	3	59.22±1.09	55.01±4.16	8.57±2.22	133.74±7.24
	4	59.04±1.05	54.77±3.80	7.93±1.84	140.17±6.67
	5	58.83±1.12	54.85±3.90	7.41±1.44	142.47±6.02
col	1	99.67±0.41	92.84±4.51	7.14±1.64	151.56±19.65
	2	99.70±0.36	92.94±4.46	7.12±1.44	187.52±21.45
	3	99.60±0.50	94.04±4.29	6.86±1.23	220.94±22.55
	4	99.33±0.56	93.98±4.39	6.79±0.93	249.89±26.61
	5	99.14±0.66	93.93±4.11	6.66±1.02	279.75±29.53
cr-a	1	91.11±0.62	85.19±3.82	5.95±0.83	111.61±7.46
	2	91.37±0.69	85.28±3.74	6.25±1.03	123.82±8.67
	3	90.84±0.96	85.36±3.90	5.59±0.93	131.69±8.00
	4	90.26±0.83	85.58±4.08	5.07±0.84	139.34±8.14
	5	89.58±0.88	85.43±3.83	4.73±0.77	152.93±9.06
gls	1	81.72±1.80	67.36±9.07	8.59±1.44	111.44±3.74
	2	81.10±1.90	69.06±8.91	6.71±1.00	146.15±4.38
	3	80.43±1.76	69.00±9.49	6.61±0.77	180.13±5.28
	4	79.91±1.95	68.06±9.91	6.43±0.61	214.56±6.93
	5	79.91±2.01	68.63±8.74	6.55±0.73	248.53±7.84
h-c1	1	94.14±0.79	79.57±6.56	8.59±1.22	64.96±5.39
	2	93.79±0.84	80.05±5.91	7.89±1.48	77.35±5.29
	3	93.03±0.86	79.40±6.22	7.25±1.21	89.32±4.76
	4	92.33±0.81	80.24±6.55	7.13±1.23	101.54±4.85
	5	91.93±1.02	80.62±6.22	7.21±1.17	113.14±4.25
h-h	1	99.66±0.33	95.39±3.70	5.78±0.61	66.77±9.17
	2	99.35±0.42	95.66±3.91	6.01±0.24	86.87±10.43
	3	99.00±0.51	95.85±3.64	6.00±0.20	103.87±10.57
	4	98.77±0.61	95.61±3.51	6.01±0.20	116.39±10.90
	5	98.61±0.57	95.95±3.42	6.00±0.40	127.15±11.85
h-s	1	93.01±0.95	79.75±7.93	7.23±1.10	36.41±2.58
	2	93.06±0.90	80.22±6.83	7.39±1.03	44.14±2.70
	3	92.35±0.85	80.59±7.13	7.12±1.01	51.87±2.69
	4	91.78±0.88	80.07±7.99	6.83±0.87	59.26±2.51
	5	91.27±0.98	79.68±7.01	6.87±0.88	66.65±2.29

Table B.1: Results of the constant learning steps strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
hep	1	99.55±0.48	87.07±7.15	5.71±0.94	19.33±1.74
	2	99.04±0.62	90.60±6.93	5.21±0.52	24.16±2.28
	3	98.53±0.77	89.12±7.06	5.12±0.40	29.55±2.36
	4	98.08±0.94	88.68±7.49	5.10±0.41	35.26±2.10
	5	97.88±1.04	89.88±7.05	5.13±0.42	39.88±2.08
ion	1	97.73±0.60	92.36±4.81	3.35±0.81	81.39±11.69
	2	96.97±0.74	92.84±5.15	2.54±0.91	85.68±7.96
	3	96.36±0.70	93.07±4.78	2.40±0.89	96.73±8.39
	4	96.00±0.65	92.78±4.80	2.21±0.74	108.72±7.92
	5	95.62±0.73	93.10±5.04	2.25±0.73	120.80±11.06
irs	1	99.08±0.64	95.11±5.74	4.47±0.73	5.16±0.24
	2	98.28±0.82	95.24±5.41	3.75±0.57	6.39±0.29
	3	97.56±1.06	94.89±5.73	3.39±0.49	7.67±0.38
	4	97.40±0.98	94.40±6.07	3.38±0.51	9.04±0.42
	5	96.78±1.11	94.31±5.88	3.29±0.47	10.31±0.43
lab	1	100.00±0.00	97.83±5.85	4.00±0.00	9.65±0.70
	2	100.00±0.00	98.17±5.18	4.00±0.00	14.55±0.84
	3	100.00±0.00	97.30±6.24	4.00±0.00	19.61±1.01
	4	100.00±0.00	97.02±7.89	3.99±0.16	24.26±1.04
	5	100.00±0.00	96.97±6.73	4.00±0.00	29.06±0.98
lym	1	98.12±0.88	80.12±10.29	9.85±2.11	37.94±2.96
	2	95.86±1.40	80.93±10.74	6.47±0.95	41.89±1.61
	3	93.83±1.56	81.68±10.19	5.50±0.86	50.00±1.76
	4	92.43±1.81	81.23±10.48	5.12±0.97	59.07±1.72
	5	91.00±1.87	80.21±10.85	4.87±0.73	69.05±2.03
pim	1	83.52±0.80	74.54±4.52	8.71±1.62	132.31±12.14
	2	83.71±0.79	74.51±4.95	8.97±2.17	145.96±11.73
	3	83.13±0.76	74.24±5.08	6.99±1.13	149.75±11.04
	4	82.68±0.78	74.80±4.35	6.43±1.22	160.04±8.46
	5	82.44±0.78	74.27±4.50	6.05±1.16	167.96±9.06
prt	1	62.37±4.32	48.07±7.49	19.97±3.93	43.58±6.94
	2	59.29±4.13	47.90±6.90	13.84±1.96	51.15±5.16
	3	57.12±3.78	47.32±6.90	11.96±1.27	59.08±4.86
	4	55.33±3.27	47.13±6.83	11.33±0.90	68.38±3.72
	5	55.63±2.82	46.64±7.32	11.32±0.75	78.84±4.55

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.1: Results of the constant learning steps strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
son	1	96.71±1.09	75.97±8.51	7.48±1.37	162.70±10.55
	2	97.01±1.21	76.15±9.37	6.93±1.00	212.03±9.20
	3	96.74±1.17	77.06±9.37	6.69±0.97	255.86±9.96
	4	96.23±1.36	76.49±9.82	6.93±1.21	301.54±10.23
	5	95.64±1.27	76.89±9.01	6.69±0.87	345.53±11.49
thy	1	99.03±0.58	92.38±5.91	6.10±0.81	10.38±0.89
	2	98.48±0.76	92.12±5.17	5.43±0.57	12.99±0.94
	3	97.91±0.78	91.36±5.41	5.20±0.49	15.84±1.01
	4	97.42±0.79	91.61±6.00	5.13±0.38	18.63±1.02
	5	97.26±0.77	91.79±5.44	5.09±0.29	21.38±1.05
vot	1	99.14±0.23	96.85±3.19	6.19±0.86	10.81±0.69
	2	98.95±0.33	96.63±3.50	5.71±0.75	10.83±0.58
	3	98.82±0.39	97.13±3.29	5.49±0.72	11.47±0.59
	4	98.49±0.57	96.65±3.31	5.12±0.77	12.22±0.69
	5	98.33±0.63	96.84±3.04	4.92±0.87	13.07±0.71
wbcd	1	98.92±0.23	95.83±2.51	4.56±0.95	24.95±2.33
	2	98.48±0.36	95.97±2.27	3.13±0.48	23.10±1.18
	3	97.91±0.50	95.96±2.28	2.64±0.55	22.41±1.24
	4	97.40±0.42	96.09±2.35	2.29±0.48	22.56±0.69
	5	97.22±0.31	96.19±2.39	2.20±0.42	23.69±0.43
wdbc	1	98.53±0.43	94.19±3.09	5.71±1.10	99.76±14.63
	2	98.30±0.45	93.85±2.87	4.57±0.85	95.15±10.46
	3	97.99±0.45	94.30±2.86	4.21±0.47	101.72±8.08
	4	97.80±0.47	94.07±3.04	4.21±0.52	109.13±7.87
	5	97.61±0.51	94.09±3.18	4.11±0.35	119.46±7.24
wine	1	99.93±0.20	92.61±5.99	4.49±0.85	21.04±1.36
	2	99.56±0.47	92.67±5.22	3.65±0.63	23.43±1.25
	3	99.17±0.59	93.71±5.26	3.43±0.62	27.57±1.31
	4	98.62±0.81	93.60±5.17	3.26±0.50	31.99±1.32
	5	98.12±1.02	92.05±5.80	3.26±0.52	36.93±1.39
wpbc	1	91.43±1.47	74.57±8.60	5.27±0.91	33.52±3.50
	2	91.77±1.36	74.67±7.84	4.38±0.61	38.37±3.59
	3	91.03±1.38	74.77±9.44	4.17±0.41	44.39±3.62
	4	90.47±1.90	75.18±9.04	4.17±0.44	51.10±4.92
	5	89.65±1.83	74.19±9.11	4.15±0.42	57.39±6.22

Table B.1: Results of the constant learning steps strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
zoo	1	99.36±0.84	93.70±7.37	8.25±1.12	4.31±0.16
	2	98.45±1.19	92.88±7.29	7.49±0.65	5.40±0.18
	3	97.18±1.43	91.01±8.11	7.13±0.56	6.61±0.23
	4	96.30±1.53	89.89±9.54	6.91±0.53	7.86±0.24
	5	95.56±1.95	89.81±10.08	6.65±0.60	9.02±0.27

B.2 Results of the tests with the constant time strategy

Table B.2: Results of the constant time strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
bal	1	87.09±0.54	79.17±4.00	11.69±1.73	57.38±5.95
	2	86.34±0.65	78.66±3.86	11.12±2.24	55.15±7.03
	3	85.82±0.63	78.49±3.80	9.70±1.62	55.18±6.59
	4	85.28±0.75	79.01±3.64	8.73±1.37	52.28±6.47
	5	84.98±0.74	78.51±3.73	8.43±1.33	52.37±5.91
bpa	1	82.08±1.58	62.50±8.60	8.96±1.45	58.66±6.47
	2	82.20±1.54	63.29±8.69	8.51±1.44	57.18±5.39
	3	81.58±1.60	63.03±7.68	7.77±1.21	57.70±4.75
	4	80.77±1.62	63.08±8.24	7.33±1.14	57.21±4.15
	5	79.85±1.63	62.07±7.51	6.95±1.02	58.23±3.33
bre	1	89.41±1.75	70.80±8.53	12.04±1.62	43.06±7.37
	2	87.69±1.81	70.27±7.40	10.71±1.44	41.11±6.55
	3	85.89±1.98	71.50±6.58	9.39±1.36	39.19±5.90
	4	84.53±2.20	71.56±7.49	8.71±1.31	37.09±5.83
	5	83.53±1.95	71.39±8.05	8.23±1.17	36.43±4.67
cmc	1	59.11±0.88	54.81±4.19	6.25±0.89	123.66±6.11
	2	59.59±1.12	54.74±4.05	10.31±3.03	123.66±8.23
	3	59.17±1.13	54.82±4.00	8.73±2.08	121.86±6.77
	4	58.75±1.08	54.94±3.75	7.69±1.68	118.76±6.44
	5	58.75±1.19	55.04±4.14	7.43±1.44	117.65±5.49

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.2: Results of the constant time strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
col	1	99.67±0.41	92.84±4.51	7.14±1.64	151.56±19.65
	2	99.69±0.38	93.04±4.28	7.24±1.50	158.14±16.53
	3	99.48±0.53	93.48±4.64	7.01±1.24	161.41±15.24
	4	99.26±0.56	93.36±4.47	7.01±1.22	164.29±16.02
	5	99.08±0.63	93.70±4.31	6.85±1.17	165.05±16.20
cr-a	1	91.11±0.62	85.19±3.82	5.95±0.83	111.61±7.46
	2	91.28±0.77	85.03±3.73	6.23±1.00	115.04±6.16
	3	90.72±0.78	85.27±3.58	5.45±0.92	114.67±6.97
	4	90.08±0.87	85.09±3.93	5.17±0.81	115.11±7.12
	5	89.48±0.89	85.27±3.99	4.60±0.73	117.37±6.84
gls	1	81.72±1.80	67.36±9.07	8.59±1.44	111.44±3.74
	2	80.64±1.93	68.30±9.27	6.85±1.04	120.28±3.86
	3	79.97±1.71	68.04±9.60	6.51±0.74	126.84±3.80
	4	79.01±2.06	67.93±8.74	6.45±0.58	132.80±4.18
	5	78.17±1.79	67.46±9.47	6.37±0.60	137.28±3.88
h-c1	1	94.14±0.79	79.57±6.56	8.59±1.22	64.96±5.39
	2	93.70±0.76	80.28±6.29	8.08±1.49	66.28±4.31
	3	92.83±0.83	79.90±5.90	7.37±1.31	67.25±3.61
	4	92.18±0.85	80.75±6.23	6.95±1.14	67.78±2.90
	5	91.70±0.85	80.36±6.59	7.12±1.14	69.01±2.55
h-h	1	99.66±0.33	95.39±3.70	5.78±0.61	66.77±9.17
	2	99.39±0.39	95.93±3.06	6.02±0.18	74.63±8.30
	3	98.99±0.56	95.97±3.29	6.00±0.20	76.50±8.20
	4	98.75±0.58	95.85±3.35	6.04±0.43	77.90±7.50
	5	98.54±0.68	95.92±3.45	6.04±0.38	77.34±7.11
h-s	1	93.01±0.95	79.75±7.93	7.23±1.10	36.41±2.58
	2	92.75±0.87	80.27±8.11	7.45±1.12	37.48±2.34
	3	92.18±0.92	80.17±7.46	7.05±1.09	38.19±2.02
	4	91.54±0.96	80.27±7.41	7.02±1.02	38.96±1.67
	5	91.17±0.88	80.44±6.65	6.87±1.01	39.38±1.45
hep	1	99.55±0.48	87.07±7.15	5.71±0.94	19.33±1.74
	2	98.99±0.72	89.28±8.25	5.19±0.55	19.41±1.79
	3	98.41±0.80	88.73±8.97	5.18±0.53	20.01±1.41
	4	98.23±0.73	88.28±7.86	5.17±0.49	20.77±1.41
	5	97.88±1.03	87.80±8.04	5.14±0.40	20.80±1.07

Table B.2: Results of the constant time strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
ion	1	97.73±0.60	92.36±4.81	3.35±0.81	81.39±11.69
	2	96.90±0.74	92.71±5.01	2.59±0.93	81.30±8.11
	3	96.35±0.64	92.84±4.73	2.39±0.89	85.85±8.13
	4	95.97±0.63	92.85±4.94	2.18±0.62	90.93±5.71
	5	95.68±0.66	92.52±5.01	2.24±0.76	96.77±8.06
irs	1	99.08±0.64	95.11±5.74	4.47±0.73	5.16±0.24
	2	98.33±0.79	95.20±5.87	3.91±0.58	5.24±0.25
	3	97.72±0.92	94.98±5.86	3.51±0.53	5.31±0.27
	4	97.22±1.03	94.58±5.68	3.34±0.47	5.38±0.23
	5	96.87±1.12	93.87±6.23	3.33±0.48	5.41±0.24
lab	1	100.00±0.00	97.83±5.85	4.00±0.00	9.65±0.70
	2	100.00±0.00	97.03±6.56	4.00±0.00	9.28±0.54
	3	100.00±0.00	98.15±5.40	4.00±0.00	9.08±0.41
	4	100.00±0.00	97.33±6.33	4.00±0.00	8.96±0.34
	5	100.00±0.00	97.92±6.00	4.00±0.00	8.91±0.30
lym	1	98.12±0.88	80.12±10.29	9.85±2.11	37.94±2.96
	2	95.89±1.32	80.80±10.82	6.69±1.11	33.11±1.39
	3	93.56±1.60	80.11±9.88	5.62±0.92	32.60±1.07
	4	91.75±1.67	81.77±10.73	5.09±0.92	32.72±0.95
	5	90.57±1.93	80.21±9.08	4.85±0.81	33.36±0.89
pim	1	83.52±0.80	74.54±4.52	8.71±1.62	132.31±12.14
	2	83.72±0.81	74.36±5.05	8.71±1.81	136.88±10.63
	3	83.12±0.79	74.27±4.56	7.15±1.47	133.37±9.56
	4	82.73±0.77	74.93±4.80	6.57±1.30	131.45±7.69
	5	82.25±0.79	74.44±4.86	6.13±0.97	132.80±6.76
prt	1	62.37±4.32	48.07±7.49	19.97±3.93	43.58±6.94
	2	59.42±4.24	48.29±7.74	13.97±2.32	43.89±4.91
	3	56.22±3.67	47.82±7.14	11.87±1.10	43.56±3.14
	4	54.87±3.64	47.24±7.01	11.39±0.85	45.20±2.75
	5	53.85±3.20	46.49±7.24	11.15±0.69	46.11±2.33
son	1	96.71±1.09	75.97±8.51	7.48±1.37	162.70±10.55
	2	97.03±1.16	77.21±8.86	7.20±1.21	171.89±7.41
	3	96.17±1.13	76.93±8.77	6.89±1.04	178.88±7.92
	4	95.69±1.39	76.92±8.38	7.05±1.07	183.40±6.23
	5	94.92±1.54	77.15±9.91	7.08±1.27	186.65±6.42

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.2: Results of the constant time strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
thy	1	99.03±0.58	92.38±5.91	6.10±0.81	10.38±0.89
	2	98.46±0.67	92.13±5.30	5.48±0.65	11.06±0.81
	3	97.77±0.82	91.84±5.58	5.17±0.40	11.74±0.74
	4	97.33±0.82	92.06±4.96	5.15±0.41	12.27±0.65
	5	97.11±0.71	92.01±5.80	5.11±0.35	12.59±0.62
vot	1	99.14±0.23	96.85±3.19	6.19±0.86	10.81±0.69
	2	98.93±0.32	96.81±3.12	5.75±0.77	10.04±0.53
	3	98.79±0.42	96.79±3.36	5.50±0.74	9.92±0.58
	4	98.54±0.54	96.93±3.30	5.33±0.81	9.89±0.54
	5	98.30±0.58	96.52±3.53	4.85±0.79	9.87±0.49
wbcd	1	98.92±0.23	95.83±2.51	4.56±0.95	24.95±2.33
	2	98.52±0.34	95.93±2.19	3.17±0.64	23.74±1.54
	3	97.95±0.48	95.92±2.37	2.63±0.56	23.48±1.19
	4	97.41±0.41	96.05±2.39	2.27±0.47	24.63±0.71
	5	97.25±0.34	96.19±2.28	2.17±0.38	26.78±0.60
wdbc	1	98.53±0.43	94.19±3.09	5.71±1.10	99.76±14.63
	2	98.31±0.44	93.82±2.94	4.72±0.82	90.12±10.58
	3	97.93±0.44	94.15±2.57	4.22±0.46	94.56±7.32
	4	97.80±0.51	93.71±2.98	4.17±0.49	98.44±7.36
	5	97.65±0.47	93.91±2.99	4.13±0.41	103.90±6.88
wine	1	99.93±0.20	92.61±5.99	4.49±0.85	21.04±1.36
	2	99.51±0.50	93.83±5.43	3.73±0.71	20.11±1.24
	3	99.01±0.71	92.92±5.69	3.41±0.57	20.62±0.87
	4	98.54±0.81	93.35±5.40	3.25±0.45	21.37±0.90
	5	98.07±1.03	92.57±6.00	3.33±0.57	22.21±0.83
wpbc	1	91.43±1.47	74.57±8.60	5.27±0.91	33.52±3.50
	2	91.38±1.52	74.79±8.72	4.41±0.64	33.67±3.55
	3	90.78±1.57	75.52±8.06	4.15±0.44	34.43±3.19
	4	89.93±1.80	75.67±8.29	4.06±0.35	35.04±3.73
	5	89.12±2.24	75.65±8.06	4.10±0.46	35.94±4.61
zoo	1	99.36±0.84	93.70±7.37	8.25±1.12	4.31±0.16
	2	98.06±1.26	91.28±8.31	7.44±0.72	3.96±0.12
	3	96.80±1.57	90.77±8.43	7.01±0.64	3.84±0.12
	4	95.31±1.95	89.21±9.10	6.75±0.67	3.75±0.10
	5	94.29±2.20	88.25±9.50	6.52±0.65	3.67±0.10

B.3 Results of the tests with the constant iterations strategy

Table B.3: Results of the constant iterations strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
bal	1	87.09±0.54	79.17±4.00	11.69±1.73	57.38±5.95
	2	86.07±0.69	78.81±4.06	10.75±2.14	29.95±3.70
	3	85.51±0.76	79.15±4.17	9.72±1.77	21.52±2.45
	4	85.04±0.86	78.46±4.02	9.11±1.61	17.54±1.96
	5	84.78±0.84	78.67±3.80	8.57±1.38	15.06±1.50
bpa	1	99.36±0.84	93.70±7.37	8.25±1.12	4.31±0.16
	2	81.21±1.54	63.07±7.95	8.60±1.45	32.66±3.19
	3	79.97±1.66	63.60±7.51	7.85±1.51	24.20±1.99
	4	79.05±1.77	62.55±6.70	7.31±1.07	20.39±1.56
	5	78.42±1.67	63.40±7.59	7.24±1.16	18.11±1.11
bre	1	82.08±1.58	62.50±8.60	8.96±1.45	58.66±6.47
	2	86.48±1.58	71.46±7.50	9.95±1.38	22.94±4.50
	3	84.81±2.08	70.51±7.70	9.04±1.51	16.46±3.00
	4	83.83±1.86	71.91±7.59	8.50±1.35	13.59±2.03
	5	82.73±1.97	71.41±7.08	8.01±1.36	11.69±1.66
cmc	1	89.41±1.75	70.80±8.53	12.04±1.62	43.06±7.37
	2	58.72±1.08	55.02±4.24	6.98±1.22	63.81±3.12
	3	58.35±0.99	54.80±3.92	6.92±1.35	44.42±2.09
	4	58.11±1.08	55.00±3.85	6.99±1.35	34.55±1.53
	5	58.09±1.06	54.41±3.92	7.28±1.39	29.11±1.26
col	1	59.11±0.88	54.81±4.19	6.25±0.89	123.66±6.11
	2	99.55±0.42	92.94±4.43	7.45±1.63	91.48±9.87
	3	99.30±0.55	92.80±4.61	7.49±1.44	70.90±6.74
	4	99.01±0.67	92.71±4.38	7.67±1.45	59.41±4.53
	5	98.76±0.93	93.33±4.26	7.53±1.35	53.21±4.38
cr-a	1	99.67±0.41	92.84±4.51	7.14±1.64	151.56±19.65
	2	90.76±0.71	85.26±4.24	6.03±1.02	61.08±3.24
	3	90.00±0.86	85.50±3.73	5.25±1.02	44.10±2.35
	4	89.50±0.98	85.33±3.91	4.98±0.93	35.31±1.88
	5	88.99±0.94	85.49±4.19	4.78±0.88	30.64±1.39

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.3: Results of the constant iterations strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
gls	1	91.11±0.62	85.19±3.82	5.95±0.83	111.61±7.46
	2	79.54±2.05	68.48±9.44	6.81±0.91	73.01±2.58
	3	77.99±1.93	67.33±8.94	6.41±0.64	59.85±1.66
	4	77.13±1.71	67.53±9.06	6.51±0.75	53.60±1.26
	5	76.33±2.05	66.57±8.93	6.56±0.73	49.93±1.07
h-c1	1	81.72±1.80	67.36±9.07	8.59±1.44	111.44±3.74
	2	93.16±0.75	80.23±5.95	8.13±1.40	38.66±2.45
	3	92.29±0.85	80.56±6.31	7.47±1.31	29.56±1.61
	4	91.68±0.70	80.83±6.34	7.45±1.30	25.26±1.18
	5	91.17±0.78	81.32±6.62	7.43±1.30	22.56±0.88
h-h	1	94.14±0.79	79.57±6.56	8.59±1.22	64.96±5.39
	2	99.28±0.44	95.60±3.57	6.05±0.29	43.25±4.74
	3	98.94±0.57	96.09±3.39	6.06±0.29	33.66±3.23
	4	98.68±0.54	95.84±3.56	6.04±0.34	28.42±2.73
	5	98.55±0.57	95.88±3.67	6.07±0.54	25.42±2.16
h-s	1	99.66±0.33	95.39±3.70	5.78±0.61	66.77±9.17
	2	92.15±0.88	80.67±7.62	7.45±1.20	22.34±1.38
	3	91.43±0.91	80.47±7.52	7.27±1.27	17.37±0.85
	4	90.86±0.89	80.59±7.05	7.07±1.10	14.92±0.64
	5	90.48±0.98	81.26±7.07	7.37±1.22	13.43±0.46
hep	1	93.01±0.95	79.75±7.93	7.23±1.10	36.41±2.58
	2	98.90±0.74	89.35±8.23	5.23±0.56	12.15±1.03
	3	98.41±0.79	87.84±8.25	5.25±0.57	9.97±0.58
	4	97.98±0.91	88.04±7.92	5.25±0.54	8.84±0.44
	5	97.63±1.05	88.09±8.79	5.40±0.71	8.18±0.33
ion	1	99.55±0.48	87.07±7.15	5.71±0.94	19.33±1.74
	2	96.68±0.74	92.77±4.90	2.50±0.70	46.00±4.88
	3	95.93±0.73	92.04±5.25	2.35±0.87	34.92±2.64
	4	95.53±0.74	92.52±5.31	2.21±0.64	29.33±1.58
	5	95.25±0.78	92.00±5.31	2.25±0.84	26.56±2.28
irs	1	97.73±0.60	92.36±4.81	3.35±0.81	81.39±11.69
	2	98.11±0.92	95.07±5.98	3.80±0.65	3.27±0.16
	3	97.51±0.94	94.80±5.65	3.41±0.53	2.64±0.11
	4	96.96±1.13	93.64±6.17	3.32±0.47	2.33±0.10
	5	96.78±1.11	94.44±5.73	3.29±0.47	2.17±0.09

Table B.3: Results of the constant iterations strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
lab	1	99.08±0.64	95.11±5.74	4.47±0.73	5.16±0.24
	2	100.00±0.00	97.02±6.77	4.00±0.00	7.55±0.39
	3	100.00±0.00	98.09±5.40	4.00±0.00	6.86±0.29
	4	100.00±0.00	97.88±5.79	4.00±0.00	6.40±0.20
	5	100.00±0.00	97.85±5.85	4.00±0.00	6.14±0.17
lym	1	100.00±0.00	97.83±5.85	4.00±0.00	9.65±0.70
	2	95.13±1.37	80.92±10.60	6.51±1.31	21.43±1.05
	3	92.79±1.64	80.33±10.48	5.72±1.13	17.11±0.52
	4	91.47±1.92	80.55±10.66	5.22±0.86	15.15±0.38
	5	89.82±2.06	80.68±9.96	4.92±0.80	14.09±0.31
pim	1	98.12±0.88	80.12±10.29	9.85±2.11	37.94±2.96
	2	82.96±0.74	74.55±4.68	8.43±1.76	70.65±5.73
	3	82.42±0.78	74.48±4.74	7.16±1.39	50.09±3.34
	4	81.85±0.79	74.73±4.61	6.85±1.27	39.66±2.47
	5	81.50±0.85	74.54±4.82	6.56±1.22	33.57±1.88
prt	1	83.52±0.80	74.54±4.52	8.71±1.62	132.31±12.14
	2	56.89±4.37	46.92±7.44	12.99±1.87	24.65±2.72
	3	54.89±3.69	45.90±6.99	11.75±1.10	19.70±1.63
	4	53.88±3.61	45.58±7.13	11.32±0.84	17.25±1.04
	5	53.26±3.00	45.78±7.16	11.14±0.74	15.81±0.84
son	1	62.37±4.32	48.07±7.49	19.97±3.93	43.58±6.94
	2	95.89±1.32	76.95±9.89	7.55±1.30	105.11±4.57
	3	95.08±1.33	75.44±9.97	7.23±1.19	85.10±3.11
	4	93.75±1.38	74.71±8.89	7.29±1.34	75.11±2.42
	5	93.22±1.66	75.06±9.85	7.39±1.21	68.61±2.09
thy	1	96.71±1.09	75.97±8.51	7.48±1.37	162.70±10.55
	2	98.09±0.73	92.11±5.19	5.47±0.64	6.67±0.41
	3	97.27±0.86	92.62±4.93	5.24±0.49	5.37±0.28
	4	97.10±0.87	91.82±5.80	5.25±0.48	4.75±0.24
	5	96.74±0.92	91.48±5.87	5.14±0.38	4.36±0.19
vot	1	99.03±0.58	92.38±5.91	6.10±0.81	10.38±0.89
	2	98.93±0.34	96.82±3.62	5.66±0.67	5.59±0.28
	3	98.65±0.45	96.85±3.24	5.35±0.82	3.98±0.21
	4	98.38±0.56	96.57±3.47	4.89±0.82	3.17±0.17
	5	98.24±0.52	96.76±3.14	4.77±0.84	2.73±0.13

APPENDIX B. FULL RESULTS OF THE EXPERIMENTATION WITH THE ILAS WINDOWING SYSTEM

Table B.3: Results of the constant iterations strategy tests of ILAS

Dataset	#Strata	Training acc.	Test acc.	#rules	Run-time (s)
wbcd	1	99.14±0.23	96.85±3.19	6.19±0.86	10.81±0.69
	2	98.38±0.35	95.73±2.61	3.23±0.53	12.38±0.61
	3	97.87±0.46	95.95±2.39	2.73±0.59	8.38±0.42
	4	97.40±0.44	96.04±2.33	2.39±0.57	6.57±0.27
	5	97.19±0.32	96.22±2.36	2.17±0.38	5.52±0.16
wdbc	1	98.92±0.23	95.83±2.51	4.56±0.95	24.95±2.33
	2	98.02±0.50	94.24±2.92	4.75±0.82	48.69±5.14
	3	97.56±0.49	94.18±2.80	4.34±0.66	34.55±2.54
	4	97.31±0.52	94.17±2.77	4.23±0.49	27.95±2.01
	5	97.20±0.53	94.01±3.17	4.22±0.47	24.14±1.59
wine	1	98.53±0.43	94.19±3.09	5.71±1.10	99.76±14.63
	2	99.48±0.53	93.35±4.79	3.78±0.68	12.49±0.65
	3	98.93±0.69	93.46±5.20	3.57±0.70	9.93±0.44
	4	98.24±1.07	93.15±5.91	3.49±0.66	8.72±0.31
	5	97.57±1.15	92.60±5.73	3.45±0.63	8.00±0.30
wpbc	1	99.93±0.20	92.61±5.99	4.49±0.85	21.04±1.36
	2	90.27±1.41	75.34±8.71	4.44±0.71	20.69±1.66
	3	89.37±1.38	76.53±9.07	4.33±0.56	16.06±1.22
	4	88.75±1.59	75.42±7.95	4.19±0.51	13.69±1.25
	5	87.29±2.48	75.95±7.90	4.20±0.60	12.39±1.30
zoo	1	91.43±1.47	74.57±8.60	5.27±0.91	33.52±3.50
	2	97.75±1.52	91.98±7.46	7.41±0.69	2.76±0.09
	3	96.04±2.20	89.56±8.67	7.03±0.68	2.25±0.10
	4	94.60±2.11	88.65±9.77	6.68±0.76	2.00±0.07
	5	93.02±2.52	87.77±9.87	6.30±0.72	1.84±0.07

Appendix C

Experimentation with generalization pressure methods

Table C.1: Results of the generalization pressure methods experimentation for the ADI and GABIL representations without ILAS

Dataset	Method	Training acc.	Test acc.	#rules
bal	MDL	86.18±0.69	80.12±4.00	9.64±1.89
	Hierar	85.01±0.73	78.88±3.92	7.05±0.99
	MOLCS-GA	89.04±0.77	81.46±3.77	15.49±1.93
bpa	MDL	78.78±1.54	61.48±8.26	7.18±1.23
	Hierar	78.12±2.15	60.85±7.44	6.54±0.94
	MOLCS-GA	83.13±2.07	61.16±8.21	12.85±2.02
bre	MDL	88.25±2.05	70.31±7.95	11.73±2.59
	Hierar	87.72±1.78	69.54±8.75	8.95±1.63
	MOLCS-GA	91.15±1.43	69.86±8.42	15.45±2.10
cmc	MDL	57.77±1.15	54.00±4.15	5.92±1.16
	Hierar	56.10±1.10	53.84±3.77	5.00±0.00
	MOLCS-GA	61.57±1.09	53.20±3.54	15.80±3.48
cr-a	MDL	89.80±0.83	84.80±4.03	5.79±1.10
	Hierar	88.60±0.76	84.83±3.94	4.39±0.62
	MOLCS-GA	91.52±0.88	84.62±4.41	10.27±1.69
h-c1	MDL	91.70±1.11	80.51±6.31	7.86±1.39
	Hierar	91.16±1.01	79.73±6.05	7.07±1.24
	MOLCS-GA	94.28±1.15	79.15±6.58	12.01±1.66

APPENDIX C. EXPERIMENTATION WITH GENERALIZATION PRESSURE METHODS

Table C.1: Results of the generalization pressure methods experimentation for the ADI and GABIL representations without ILAS

Dataset	Method	Training acc.	Test acc.	#rules
h-h	MDL	99.41±0.41	95.71±3.70	6.13±0.41
	Hierar	98.54±0.52	95.79±3.57	6.01±0.08
	MOLCS-GA	99.38±0.44	94.70±4.40	6.17±0.43
h-s	MDL	90.02±1.06	79.46±7.35	6.09±0.31
	Hierar	90.04±1.07	79.85±7.86	6.51±0.92
	MOLCS-GA	92.51±1.26	78.69±7.57	10.55±2.08
hep	MDL	98.76±0.78	88.34±7.63	5.49±0.78
	Hierar	99.05±0.62	88.30±7.80	5.93±1.25
	MOLCS-GA	98.86±0.71	88.51±8.39	6.08±1.44
ion	MDL	96.02±0.71	92.40±4.96	2.09±0.33
	Hierar	95.32±0.77	91.54±5.22	2.15±0.43
	MOLCS-GA	97.49±0.84	90.07±5.75	5.54±1.80
irs	MDL	98.84±0.74	95.33±5.85	4.26±0.89
	Hierar	98.80±0.77	94.13±6.57	4.27±0.91
	MOLCS-GA	98.73±0.74	94.18±6.32	4.45±0.88
lab	MDL	100.00±0.00	95.90±8.52	4.00±0.00
	Hierar	100.00±0.00	94.46±9.65	4.00±0.00
	MOLCS-GA	100.00±0.00	98.28±5.47	4.08±0.29
lrn	MDL	76.47±1.39	68.07±4.99	9.43±1.42
	Hierar	74.77±1.58	68.74±5.08	6.25±0.61
	MOLCS-GA	79.84±1.96	65.84±4.33	16.31±2.54
lym	MDL	97.69±1.47	78.10±10.94	10.63±2.10
	Hierar	98.50±1.05	76.79±11.49	12.93±2.72
	MOLCS-GA	97.50±1.26	76.76±11.19	9.76±1.59
mmg	MDL	81.95±1.80	65.89±10.59	6.36±0.64
	Hierar	80.90±1.57	66.48±10.14	6.21±0.69
	MOLCS-GA	85.50±1.87	64.92±10.13	8.91±1.63
pim	MDL	81.72±0.89	73.99±4.93	6.59±1.33
	Hierar	79.59±1.05	73.97±4.95	5.07±0.30
	MOLCS-GA	84.63±1.15	74.12±4.56	12.66±2.34
prt	MDL	58.91±4.70	47.83±8.11	15.97±3.75
	Hierar	55.97±3.46	46.15±7.04	10.10±0.38
	MOLCS-GA	61.60±4.68	46.92±7.40	19.52±3.47
thy	MDL	98.38±0.73	91.90±5.78	5.77±0.87
	Hierar	97.65±1.29	91.64±5.62	5.55±0.74
	MOLCS-GA	98.39±0.71	91.90±5.95	6.17±1.07

Table C.1: Results of the generalization pressure methods experimentation for the ADI and GABIL representations without ILAS

Dataset	Method	Training acc.	Test acc.	#rules
vot	MDL	98.91±0.44	96.53±3.34	6.08±1.12
	Hierar	98.29±0.52	96.39±3.25	4.41±0.72
	MOLCS-GA	98.99±0.40	96.39±3.34	6.57±1.43
wbcd	MDL	98.06±0.35	95.89±2.51	3.36±0.90
	Hierar	96.94±0.39	95.81±2.54	2.05±0.24
	MOLCS-GA	98.72±0.33	95.65±2.46	5.73±1.69
wdbc	MDL	97.38±0.69	94.09±2.91	4.68±1.14
	Hierar	96.23±0.74	94.13±2.81	3.41±0.79
	MOLCS-GA	98.24±0.55	93.98±3.14	7.00±1.40
wine	MDL	99.79±0.32	94.01±5.21	4.59±0.99
	Hierar	99.75±0.36	93.02±5.57	4.56±1.07
	MOLCS-GA	99.84±0.30	91.84±6.13	5.60±1.19
wpbc	MDL	86.15±2.12	75.50±7.35	3.05±0.94
	Hierar	85.51±2.01	75.98±7.55	2.77±1.05
	MOLCS-GA	89.98±2.33	71.58±8.37	6.97±1.87
zoo	MDL	98.56±2.18	93.67±7.99	7.65±1.04
	Hierar	99.37±1.40	91.69±7.96	7.83±0.96
	MOLCS-GA	98.54±2.18	90.31±8.69	8.03±1.14

Table C.2: Results of the generalization pressure methods experimentation for the ADI and GABIL representations using ILAS with 2 strata

Dataset	Method	Training acc.	Test acc.	#rules
bal	MDL	85.36±0.75	79.43±3.88	8.65±1.80
	Hierar	84.42±0.83	78.58±4.06	6.41±0.74
	MOLCS-GA	86.02±1.07	80.23±4.33	11.75±1.65
bpa	MDL	77.71±1.50	61.32±7.72	6.67±0.94
	Hierar	76.55±1.94	61.75±8.16	6.16±0.52
	MOLCS-GA	75.11±2.82	59.81±9.11	9.07±1.82
bre	MDL	83.95±1.98	71.89±7.50	7.49±1.36
	Hierar	86.95±1.79	69.40±8.02	8.99±1.62
	MOLCS-GA	84.36±1.96	68.46±8.56	9.87±2.20
cmc	MDL	57.27±0.84	54.26±3.92	5.87±1.06
	Hierar	56.13±1.25	53.96±3.84	5.00±0.00
	MOLCS-GA	57.68±1.41	52.08±4.40	10.90±2.23
cr-a	MDL	89.30±0.89	84.52±3.96	5.66±1.25
	Hierar	88.42±0.80	84.65±4.00	4.21±0.47
	MOLCS-GA	88.99±0.98	83.72±4.46	8.19±1.43
h-c1	MDL	90.58±1.06	80.11±5.51	7.13±1.32
	Hierar	90.41±1.07	79.86±5.69	6.83±1.16
	MOLCS-GA	88.64±1.67	78.79±7.63	8.67±1.86
h-h	MDL	98.98±0.55	95.85±3.56	6.07±0.29
	Hierar	98.90±0.55	95.77±3.57	6.00±0.00
	MOLCS-GA	98.11±0.78	95.97±3.69	6.07±0.57
h-s	MDL	89.25±1.15	80.10±7.31	6.55±0.82
	Hierar	89.38±0.98	80.42±7.17	6.53±0.93
	MOLCS-GA	87.22±1.95	77.33±8.54	7.56±1.53
hep	MDL	97.60±1.10	89.75±7.30	5.15±0.50
	Hierar	97.78±1.04	88.59±7.26	5.21±0.57
	MOLCS-GA	95.41±1.90	87.99±7.91	5.34±0.68
ion	MDL	94.80±8.05	91.56±9.82	2.13±0.64
	Hierar	95.67±0.77	92.47±5.05	2.07±0.29
	MOLCS-GA	95.01±1.19	90.42±5.77	3.83±1.29
irs	MDL	97.42±1.16	94.80±5.60	3.34±0.51
	Hierar	97.61±0.97	95.47±5.41	3.29±0.50
	MOLCS-GA	97.72±0.97	94.84±6.12	3.69±0.75
lab	MDL	100.00±0.00	96.49±7.99	4.00±0.00
	Hierar	100.00±0.00	96.77±7.32	4.01±0.08
	MOLCS-GA	99.77±0.78	96.63±7.32	3.90±0.41

Table C.2: Results of the generalization pressure methods experimentation for the ADI and GABIL representations using ILAS with 2 strata

Dataset	Method	Training acc.	Test acc.	#rules
lrn	MDL	75.75±1.16	68.29±4.86	7.75±1.39
	Hierar	74.29±1.72	68.70±4.89	6.05±0.24
	MOLCS-GA	74.29±2.08	64.68±4.76	12.47±1.93
lym	MDL	94.62±2.07	79.27±12.02	6.75±1.60
	Hierar	94.72±1.84	77.29±10.40	6.97±1.64
	MOLCS-GA	91.43±2.47	75.63±11.88	7.19±1.46
mmg	MDL	80.44±1.58	68.50±10.21	6.48±0.75
	Hierar	80.03±1.63	67.20±10.53	6.21±0.55
	MOLCS-GA	77.42±2.81	63.97±10.72	6.58±0.93
pim	MDL	81.03±0.88	74.32±4.69	5.67±0.91
	Hierar	79.19±1.00	74.24±5.04	5.02±0.14
	MOLCS-GA	80.28±1.17	72.69±5.40	9.19±1.82
prt	MDL	54.35±4.57	45.77±6.80	11.41±1.56
	Hierar	54.22±3.72	44.95±6.61	10.05±0.43
	MOLCS-GA	54.46±4.12	44.19±8.12	13.90±3.07
thy	MDL	97.16±0.90	92.00±5.31	5.14±0.40
	Hierar	97.18±1.00	91.82±6.19	5.14±0.38
	MOLCS-GA	95.87±1.29	91.41±6.10	5.27±0.62
vot	MDL	98.13±0.60	96.32±3.66	4.73±0.88
	Hierar	97.93±0.62	96.37±3.16	4.13±0.41
	MOLCS-GA	97.92±0.58	95.95±3.03	4.99±0.96
wbcd	MDL	97.51±0.38	96.32±2.57	2.21±0.51
	Hierar	97.13±0.37	96.04±2.51	2.00±0.00
	MOLCS-GA	97.77±0.49	95.43±2.52	3.83±0.94
wdbc	MDL	96.62±0.79	93.69±3.04	3.56±0.91
	Hierar	95.60±0.80	93.45±3.34	2.85±0.64
	MOLCS-GA	96.64±0.82	93.32±3.36	5.05±1.16
wine	MDL	98.92±0.78	92.81±6.06	3.69±0.77
	Hierar	98.98±0.75	92.88±5.64	3.60±0.66
	MOLCS-GA	97.78±1.42	90.48±6.98	4.51±1.08
wpbc	MDL	83.55±2.88	75.70±7.38	2.52±0.76
	Hierar	84.31±2.55	75.81±7.21	2.35±0.62
	MOLCS-GA	84.12±2.18	72.46±8.41	4.72±1.35
zoo	MDL	96.42±2.96	90.66±8.54	6.85±0.93
	Hierar	98.41±1.97	90.38±8.69	7.36±0.84
	MOLCS-GA	95.91±2.63	87.81±10.24	7.23±1.04

Table C.3: Results of the experimentation with generalization pressure methods for the UBR and XCS representations without using ILAS

Dataset	Method	Training acc.	Test acc.	#rules
bal	MDL	89.55±0.89	82.09±3.62	15.81±2.25
	Hierar	86.86±1.03	80.57±4.03	8.23±1.26
	MOLCS-GA	90.28±0.79	81.70±3.99	18.08±2.31
bpa	MDL	81.15±2.41	65.55±7.79	10.38±2.56
	Hierar	82.30±2.53	63.95±8.38	8.36±2.16
	MOLCS-GA	85.19±2.21	64.21±8.04	14.61±2.93
bre	MDL	83.11±1.92	71.83±6.95	12.63±2.79
	Hierar	81.80±2.05	71.80±6.71	8.72±2.94
	MOLCS-GA	82.00±2.39	71.99±7.08	9.98±3.18
cmc	MDL	58.61±1.34	54.22±4.05	10.11±2.36
	Hierar	58.20±1.32	54.46±3.83	5.03±0.21
	MOLCS-GA	61.66±1.28	54.07±3.79	16.44±3.63
cr-a	MDL	89.22±1.10	84.79±3.95	8.83±2.29
	Hierar	87.96±0.93	84.79±3.79	4.27±0.51
	MOLCS-GA	89.45±1.31	84.71±4.04	8.55±2.31
h-c1	MDL	92.37±1.19	78.79±6.96	10.81±1.79
	Hierar	92.13±1.29	79.16±6.77	9.37±2.05
	MOLCS-GA	93.66±1.28	79.49±6.59	13.08±2.16
h-h	MDL	99.04±0.53	95.86±3.33	6.26±0.62
	Hierar	98.89±0.44	95.51±3.47	6.01±0.08
	MOLCS-GA	98.86±0.62	95.95±3.78	6.15±0.43
h-s	MDL	91.10±1.65	79.98±7.53	8.33±1.53
	Hierar	92.78±1.47	78.12±7.40	12.75±3.39
	MOLCS-GA	92.99±1.45	79.21±7.50	12.98±2.22
hep	MDL	98.31±0.93	87.38±7.79	6.91±1.58
	Hierar	98.76±0.81	86.04±8.85	7.39±1.73
	MOLCS-GA	98.34±0.98	87.69±8.36	6.25±1.41
ion	MDL	96.41±1.01	90.30±5.07	4.26±1.20
	Hierar	91.43±5.71	79.02±10.20	21.13±7.45
	MOLCS-GA	94.80±4.70	83.76±8.95	17.29±7.41
irs	MDL	98.17±0.81	94.27±6.16	3.74±0.74
	Hierar	98.54±0.88	93.56±6.32	4.11±0.90
	MOLCS-GA	98.37±0.80	94.49±6.29	3.78±0.84
lab	MDL	100.00±0.00	97.28±6.30	4.00±0.00
	Hierar	100.00±0.00	97.21±7.36	4.00±0.00
	MOLCS-GA	100.00±0.00	97.97±5.47	4.09±0.29

Table C.3: Results of the experimentation with generalization pressure methods for the UBR and XCS representations without using ILAS

Dataset	Method	Training acc.	Test acc.	#rules
lrn	MDL	78.17±1.87	67.95±5.13	14.83±2.48
	Hierar	76.51±1.81	69.84±5.37	6.62±0.82
	MOLCS-GA	80.41±1.84	68.39±4.40	17.44±3.02
lym	MDL	94.51±2.28	78.22±10.67	9.85±1.86
	Hierar	95.51±2.12	79.09±10.51	10.29±2.06
	MOLCS-GA	94.43±2.34	78.72±11.51	9.22±1.66
mmg	MDL	82.38±2.39	63.21±10.16	7.39±1.31
	Hierar	84.19±2.32	63.08±9.56	8.93±2.31
	MOLCS-GA	84.97±2.42	62.46±9.98	11.09±2.40
pim	MDL	83.25±1.56	74.70±4.91	11.23±2.92
	Hierar	82.49±1.15	73.57±5.12	5.58±0.92
	MOLCS-GA	85.57±1.57	73.51±4.54	17.40±4.35
prt	MDL	57.11±4.23	46.15±6.55	16.25±3.06
	Hierar	53.17±3.81	45.54±7.44	10.33±0.96
	MOLCS-GA	46.63±11.93	40.99±10.59	10.53±2.83
thy	MDL	97.99±1.05	92.41±4.75	6.21±0.96
	Hierar	98.19±1.09	92.88±5.22	5.90±0.96
	MOLCS-GA	98.17±0.92	91.56±5.43	6.05±1.09
vot	MDL	98.87±0.41	96.38±3.37	6.06±1.11
	Hierar	98.38±0.52	96.39±3.24	4.40±0.66
	MOLCS-GA	98.95±0.34	96.02±3.49	6.33±1.27
wbcd	MDL	98.54±0.32	95.70±2.58	6.81±1.43
	Hierar	97.83±0.41	95.63±2.54	3.25±0.91
	MOLCS-GA	98.66±0.36	95.61±2.62	7.59±1.81
wdbc	MDL	96.88±0.80	94.05±2.82	4.40±0.97
	Hierar	97.25±0.66	92.00±3.54	7.61±2.59
	MOLCS-GA	97.76±0.65	92.59±3.45	8.97±2.43
wine	MDL	99.86±0.28	91.96±6.46	6.33±1.45
	Hierar	99.91±0.24	90.37±7.35	6.76±1.64
	MOLCS-GA	99.84±0.31	91.52±7.11	5.79±1.43
wpbc	MDL	83.64±3.71	73.80±6.86	4.37±1.40
	Hierar	89.24±3.49	65.50±9.44	16.27±6.57
	MOLCS-GA	87.09±3.57	69.55±8.49	10.05±3.72
zoo	MDL	97.83±2.37	92.47±8.62	7.83±1.24
	Hierar	98.19±1.95	91.74±8.11	7.75±1.15
	MOLCS-GA	96.11±3.00	89.96±8.90	6.87±1.06

Table C.4: Results of the experimentation with generalization pressure methods for the UBR and XCS representations without using ILAS

Dataset	Method	Training acc.	Test acc.	#rules
bal	MDL	87.50±1.29	80.85±4.04	11.77±3.00
	Hierar	86.03±1.27	80.87±3.54	7.67±1.21
	MOLCS-GA	87.32±1.05	81.34±4.08	12.77±1.83
bpa	MDL	77.27±2.41	63.64±7.92	7.37±1.35
	Hierar	79.94±2.57	63.67±7.63	7.46±1.72
	MOLCS-GA	77.81±2.69	62.54±7.61	10.02±2.16
bre	MDL	80.89±1.87	72.44±6.74	8.90±2.22
	Hierar	81.53±1.78	72.69±7.78	9.56±2.21
	MOLCS-GA	78.17±1.58	71.57±6.45	5.84±1.32
cmc	MDL	57.49±1.41	54.18±4.03	6.78±1.35
	Hierar	57.84±1.14	54.43±4.12	5.00±0.00
	MOLCS-GA	58.56±1.35	54.21±4.20	10.57±2.25
cr-a	MDL	87.93±1.00	84.69±4.08	6.11±1.46
	Hierar	87.56±0.73	84.56±4.03	4.12±0.34
	MOLCS-GA	87.66±0.98	84.63±4.08	5.81±1.62
h-c1	MDL	89.93±1.35	79.65±7.04	7.92±1.41
	Hierar	91.19±1.30	79.24±6.85	8.49±1.78
	MOLCS-GA	89.19±1.71	77.97±7.17	8.93±1.74
h-h	MDL	98.57±0.49	96.49±3.33	6.13±0.46
	Hierar	98.84±0.52	95.27±4.44	6.01±0.08
	MOLCS-GA	97.82±0.91	95.72±3.81	6.07±0.66
h-s	MDL	89.14±1.86	79.53±7.51	7.60±1.48
	Hierar	91.74±1.61	77.98±8.02	11.05±2.82
	MOLCS-GA	87.93±2.09	77.80±8.11	8.83±1.99
hep	MDL	96.94±1.27	87.54±8.07	5.61±0.88
	Hierar	97.69±1.13	85.82±9.15	5.79±0.88
	MOLCS-GA	95.33±2.25	86.98±9.03	5.18±0.63
ion	MDL	95.15±1.13	90.34±5.71	3.37±1.15
	Hierar	93.51±5.02	84.50±8.41	9.84±4.62
	MOLCS-GA	89.28±7.30	81.68±10.50	8.12±2.82
irs	MDL	97.36±1.01	94.44±5.83	3.28±0.49
	Hierar	97.91±0.82	94.31±6.32	3.43±0.62
	MOLCS-GA	97.79±0.84	94.67±6.11	3.34±0.60
lab	MDL	100.00±0.00	97.43±6.54	4.01±0.08
	Hierar	100.00±0.00	95.53±9.35	4.00±0.00
	MOLCS-GA	99.80±0.63	97.89±5.82	3.65±0.62

Table C.4: Results of the experimentation with generalization pressure methods for the UBR and XCS representations without using ILAS

Dataset	Method	Training acc.	Test acc.	#rules
lrn	MDL	75.67±2.20	68.35±4.82	8.76±1.69
	Hierar	74.68±2.19	68.29±5.59	6.23±0.51
	MOLCS-GA	75.62±1.61	67.59±4.63	12.29±1.88
lym	MDL	89.91±2.88	78.13±10.63	6.85±1.37
	Hierar	91.70±2.76	77.87±10.58	7.25±1.36
	MOLCS-GA	89.24±2.64	75.95±11.54	6.53±1.37
mmg	MDL	80.59±2.23	63.27±10.26	6.74±0.90
	Hierar	81.65±2.51	63.44±10.42	7.57±1.67
	MOLCS-GA	78.32±2.74	61.50±10.50	7.53±1.34
pim	MDL	80.59±1.46	74.46±4.51	6.91±1.81
	Hierar	81.07±1.20	74.37±4.54	5.13±0.54
	MOLCS-GA	81.63±1.30	73.52±4.73	10.99±2.43
prt	MDL	55.58±4.08	45.37±7.69	14.57±2.40
	Hierar	53.24±3.66	45.68±7.09	10.39±0.92
	MOLCS-GA	44.46±8.06	40.55±8.24	8.68±2.12
thy	MDL	96.71±1.32	92.06±5.13	5.46±0.66
	Hierar	97.34±1.26	92.02±5.25	5.57±0.74
	MOLCS-GA	96.46±1.51	91.58±6.00	5.39±0.77
vot	MDL	98.20±0.70	96.66±2.97	4.87±0.94
	Hierar	97.90±0.66	96.38±3.33	4.29±0.69
	MOLCS-GA	97.94±0.64	96.00±3.33	5.02±1.10
wbcd	MDL	98.17±0.45	95.60±2.48	5.45±1.61
	Hierar	97.79±0.44	95.81±2.50	3.22±0.92
	MOLCS-GA	97.86±0.48	95.53±2.28	5.02±1.29
wdbc	MDL	96.45±0.87	93.64±3.01	4.37±1.05
	Hierar	96.79±0.81	92.26±3.45	5.52±1.59
	MOLCS-GA	96.19±0.88	91.85±3.73	5.89±1.60
wine	MDL	99.45±0.56	92.55±6.19	5.27±1.25
	Hierar	99.71±0.42	91.18±7.63	5.74±1.38
	MOLCS-GA	98.92±0.82	92.25±6.45	4.32±1.12
wpbc	MDL	79.84±2.81	75.55±5.47	2.93±0.87
	Hierar	85.40±3.69	68.13±8.70	9.02±3.64
	MOLCS-GA	81.32±2.97	69.85±8.88	5.39±1.93
zoo	MDL	95.97±3.40	90.94±9.38	7.00±1.07
	Hierar	97.35±2.18	91.35±9.46	7.27±0.90
	MOLCS-GA	94.39±3.07	88.26±9.72	6.60±0.96

APPENDIX C. EXPERIMENTATION WITH GENERALIZATION PRESSURE METHODS

Appendix D

Full results of the global comparison with alternative machine learning systems

D.1 Results on small datasets

Table D.1: Results of global comparative tests on the *aud* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	81.01±5.29	70.62±8.72	13.93±1.51
GAssist-gr2	81.01±5.29	70.62±8.72	13.93±1.51
GAssist-gr3	81.01±5.29	70.62±8.72	13.93±1.51
GAssist-inst	81.01±5.29	70.62±8.72	13.93±1.51
Majority	25.22±0.34	25.29±2.89	
C4.5	90.20±0.90	77.18±8.18	30.83±1.90
PART	90.74±0.90	79.86±8.28	19.87±1.65
IB1	100.00±0.00	74.01±9.37	
IBk	78.29±1.36	69.56±9.91	
NB-gaussian	78.55±1.13	71.35±7.98	
NB-kernel	78.55±1.13	71.35±7.98	
LIBSVM	76.53±1.50	69.17±8.06	193.60±2.74
XCS	89.84±3.72	79.6±12.3	

APPENDIX D. FULL RESULTS OF THE GLOBAL COMPARISON WITH ALTERNATIVE MACHINE LEARNING SYSTEMS

Table D.2: Results of global comparative tests on the *aut* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	81.04±2.55	66.17±10.65	7.83±1.09
GAssist-gr2	81.77±2.66	67.91±10.18	7.71±1.06
GAssist-gr3	81.63±2.35	67.02±10.13	8.11±1.22
GAssist-inst	93.07±1.56	65.98±9.65	37.38±4.42
Majority	32.68±0.32	32.68±2.81	
C4.5	93.91±1.33	80.57±8.73	46.90±6.93
PART	92.92±1.77	75.42±9.81	20.57±2.62
IB1	98.57±0.59	73.92±8.35	
IBk	85.24±1.21	68.87±7.22	
NB-gaussian	67.43±1.84	56.82±9.99	
NB-kernel	75.03±1.73	61.45±8.57	
LIBSVM	66.89±2.29	57.56±8.36	175.07±2.46
XCS	99.60±0.46	71.2±9.9	

Table D.3: Results of global comparative tests on the *bal* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	84.90±0.86	79.27±3.98	9.39±1.53
GAssist-gr2	84.76±0.88	79.27±4.16	8.94±1.50
GAssist-gr3	86.34±0.65	78.66±3.86	11.12±2.24
GAssist-inst	92.14±0.28	89.62±2.22	37.59±16.98
Majority	46.18±0.10	45.24±0.89	
C4.5	89.93±0.68	77.66±2.91	42.33±4.69
PART	93.39±0.98	83.22±4.59	38.97±3.23
IB1	100.00±0.00	77.42±5.47	
IBk	90.53±0.54	86.09±2.72	
NB-gaussian	90.70±0.26	90.26±1.71	
NB-kernel	91.92±0.25	91.43±1.25	
LIBSVM	91.01±0.19	90.90±1.43	174.03±3.18
XCS	95.19±1.28	81.1±3.8	

Table D.4: Results of global comparative tests on the *bpa* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	81.62±1.52	64.91±6.85	8.29±1.56
GAssist-gr2	78.66±1.57	64.03±7.24	7.71±1.46
GAssist-gr3	82.20±1.54	63.29±8.69	8.51±1.44
GAssist-inst	83.14±1.79	66.64±8.09	27.58±8.31
Majority	57.97±0.13	57.99±1.11	
C4.5	86.29±3.90	65.70±7.91	25.70±5.69
PART	77.10±4.96	64.73±6.83	7.67±2.43
IB1	100.00±0.00	61.90±9.69	
IBk	82.62±1.37	60.55±8.35	
NB-gaussian	56.82±2.52	55.49±8.77	
NB-kernel	73.47±1.25	65.15±9.07	
LIBSVM	59.43±0.91	58.37±1.82	264.93±1.71
XCS	99.97±0.10	67.1±7.5	

Table D.5: Results of global comparative tests on the *bps* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	87.96±0.62	81.54±3.82	7.47±1.49
GAssist-gr2	88.09±0.65	81.97±3.62	7.25±1.48
GAssist-gr3	88.03±0.68	81.76±3.60	7.57±1.52
GAssist-inst	89.97±0.73	84.13±3.77	18.71±7.09
Majority	51.61±0.05	51.61±0.41	
C4.5	96.26±1.09	80.30±3.75	50.80±5.10
PART	90.25±2.21	80.73±3.37	13.20±2.70
IB1	100.00±0.00	82.90±3.65	
IBk	91.38±0.52	83.91±3.43	
NB-gaussian	78.68±0.34	78.42±2.86	
NB-kernel	80.72±0.39	79.52±3.98	
LIBSVM	86.63±0.43	85.66±3.71	441.10±5.23

Table D.6: Results of global comparative tests on the *bre* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	87.69±1.81	70.27±7.40	10.71±1.44
GAssist-gr2	87.69±1.81	70.27±7.40	10.71±1.44
GAssist-gr3	87.69±1.81	70.27±7.40	10.71±1.44
GAssist-inst	87.69±1.81	70.27±7.40	10.71±1.44
Majority	70.28±0.17	70.31±1.50	
C4.5	76.83±1.35	73.91±5.83	9.33±9.61
PART	80.78±1.85	67.32±7.13	17.30±4.62
IB1	97.95±0.41	69.84±7.73	
IBk	80.02±0.99	73.89±4.24	
NB-gaussian	75.43±1.04	72.14±8.18	
NB-kernel	75.43±1.04	72.14±8.18	
LIBSVM	79.28±1.09	74.35±4.70	172.87±3.84
XCS	93.84±1.20	70.1±8.0	

Table D.7: Results of global comparative tests on the *cmc* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	58.70±0.98	54.63±3.68	7.01±1.42
GAssist-gr2	58.68±1.04	54.93±4.12	6.89±1.20
GAssist-gr3	59.59±1.12	54.74±4.05	10.31±3.03
GAssist-inst	56.99±1.70	52.54±3.96	14.52±5.45
Majority	42.70±0.04	42.70±0.33	
C4.5	71.08±1.44	51.96±4.23	148.00±21.61
PART	75.77±1.00	51.32±4.21	165.77±10.59
IB1	95.57±0.27	43.72±4.41	
IBk	69.54±0.59	46.55±5.20	
NB-gaussian	51.71±0.63	50.77±3.22	
NB-kernel	53.15±0.60	51.69±3.19	
LIBSVM	54.79±0.70	48.06±3.15	1228.57±5.70
XCS	71.22±2.34	52.4±3.6	

APPENDIX D. FULL RESULTS OF THE GLOBAL COMPARISON WITH ALTERNATIVE MACHINE LEARNING SYSTEMS

Table D.8: Results of global comparative tests on the *col* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	99.38±0.59	93.39±4.83	6.88±1.38
GAssist-gr2	99.31±0.58	92.84±4.50	7.84±1.61
GAssist-gr3	99.69±0.38	93.04±4.28	7.24±1.50
GAssist-inst	99.02±0.55	89.43±4.89	21.99±9.19
Majority	63.04±0.17	63.07±1.52	
C4.5	86.52±0.87	85.50±4.80	5.97±2.01
PART	87.18±0.90	84.23±4.80	9.30±3.10
IB1	98.98±0.26	79.10±6.52	
IBk	87.52±0.68	81.45±5.79	
NB-gaussian	79.68±0.81	78.50±6.39	
NB-kernel	80.53±0.71	79.14±6.00	
LIBSVM	89.61±0.44	84.89±4.77	196.27±5.59
XCS	94.25±1.21	84.0±5.8	

Table D.9: Results of global comparative tests on the *cr-a* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	91.07±0.73	85.62±4.00	5.78±1.08
GAssist-gr2	90.87±0.68	85.56±3.88	5.67±1.00
GAssist-gr3	91.28±0.77	85.03±3.73	6.23±1.00
GAssist-inst	90.61±0.61	85.18±3.82	18.27±7.61
Majority	55.51±0.08	55.51±0.70	
C4.5	90.31±0.86	85.55±3.45	22.63±8.58
PART	93.23±0.82	84.23±3.98	31.00±6.58
IB1	99.47±0.11	81.92±4.41	
IBk	91.05±0.52	84.73±4.04	
NB-gaussian	78.36±0.63	77.55±4.76	
NB-kernel	82.58±0.82	81.07±5.32	
LIBSVM	55.51±0.08	55.51±0.70	553.60±1.65
XCS	98.90±0.73	85.6±3.5	

Table D.10: Results of global comparative tests on the *cr-g* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	82.87±0.90	72.18±3.60	12.45±2.48
GAssist-gr2	82.61±0.91	72.17±4.05	11.99±2.22
GAssist-gr3	82.74±0.92	72.29±4.13	12.69±2.02
GAssist-inst	82.13±0.95	72.46±3.98	28.43±11.95
Majority	70.00±0.00	70.00±0.00	
C4.5	85.10±1.77	70.93±3.35	87.93±17.63
PART	90.38±0.97	70.60±4.04	68.13±6.33
IB1	100.00±0.00	72.23±3.77	
IBk	85.74±0.54	72.47±3.77	
NB-gaussian	77.01±0.45	74.80±3.65	
NB-kernel	77.37±0.43	74.07±3.86	
LIBSVM	82.69±0.58	75.90±3.48	567.27±5.88
XCS	97.92±0.81	70.9±4.3	

Table D.11: Results of global comparative tests on the *gls* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	79.78±2.07	68.46±8.83	6.61±0.77
GAssist-gr2	79.20±2.15	67.68±10.00	6.69±0.78
GAssist-gr3	80.64±1.93	68.30±9.27	6.85±1.04
GAssist-inst	87.89±2.10	68.28±9.96	32.10±5.36
Majority	35.52±0.44	35.68±3.72	
C4.5	92.91±1.66	68.81±9.56	23.70±2.76
PART	93.08±1.70	69.56±9.17	15.40±1.96
IB1	100.00±0.00	69.72±8.31	
IBk	81.12±1.35	70.36±8.44	
NB-gaussian	55.81±1.92	48.94±9.72	
NB-kernel	57.72±1.88	51.00±7.78	
LIBSVM	61.23±2.77	58.89±11.53	178.93±2.84
XCS	97.62±1.37	71.8±8.9	

Table D.12: Results of global comparative tests on the *h-c1* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	92.69±0.84	80.34±6.53	7.41±1.29
GAssist-gr2	92.25±0.83	81.18±6.41	7.35±1.14
GAssist-gr3	93.70±0.76	80.28±6.29	8.08±1.49
GAssist-inst	92.23±0.97	81.55±6.68	23.93±8.49
Majority	54.46±0.19	54.47±1.65	
C4.5	91.80±1.26	77.55±6.32	27.63±4.48
PART	94.54±1.26	80.32±8.11	19.63±3.09
IB1	100.00±0.00	76.55±5.92	
IBk	88.61±0.72	81.48±6.72	
NB-gaussian	84.56±0.72	83.63±7.31	
NB-kernel	85.93±0.84	84.52±6.68	
LIBSVM	88.82±0.70	82.99±5.58	136.73±3.88
XCS	99.81±0.31	76.5±7.9	

Table D.13: Results of global comparative tests on the *h-h* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	99.32±0.44	95.77±3.09	5.99±0.16
GAssist-gr2	99.20±0.46	95.88±3.52	6.01±0.23
GAssist-gr3	99.39±0.39	95.93±3.06	6.02±0.18
GAssist-inst	99.68±0.24	95.57±3.82	9.76±4.01
Majority	63.95±0.21	63.95±1.86	
C4.5	84.26±2.32	79.51±7.38	7.07±4.42
PART	85.74±1.45	80.38±5.60	8.53±2.58
IB1	99.31±0.26	77.76±8.38	
IBk	90.27±0.77	81.52±6.13	
NB-gaussian	85.59±0.70	84.56±6.03	
NB-kernel	85.68±0.76	85.25±5.87	
LIBSVM	86.72±0.51	81.78±7.61	113.83±3.24
XCS	98.51±0.96	77.8±8.0	

APPENDIX D. FULL RESULTS OF THE GLOBAL COMPARISON WITH ALTERNATIVE MACHINE LEARNING SYSTEMS

Table D.14: Results of global comparative tests on the *h-s* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	91.30±0.92	81.14±7.32	7.05±1.13
GAssist-gr2	90.88±0.97	80.77±7.25	6.88±1.06
GAssist-gr3	92.75±0.87	80.27±8.11	7.45±1.12
GAssist-inst	92.06±1.13	81.21±7.05	14.50±5.01
Majority	55.56±0.00	55.56±0.00	
C4.5	92.19±1.70	79.75±6.54	16.97±3.38
PART	95.73±1.15	78.02±6.55	17.70±2.04
IB1	100.00±0.00	76.54±8.24	
IBk	89.62±0.83	79.14±6.73	
NB-gaussian	85.69±1.20	84.32±7.93	
NB-kernel	86.43±0.87	84.20±6.96	
LIBSVM	86.63±1.01	82.72±7.18	120.23±3.94
XCS	99.85±0.24	75.3±8.1	

Table D.15: Results of global comparative tests on the *hep* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.82±0.65	88.06±7.59	5.31±0.68
GAssist-gr2	98.58±0.75	87.20±8.47	5.31±0.57
GAssist-gr3	98.99±0.72	89.28±8.25	5.19±0.55
GAssist-inst	97.18±1.10	83.83±8.32	9.94±4.04
Majority	79.36±0.25	79.38±2.15	
C4.5	92.19±2.68	78.24±7.75	8.47±2.94
PART	94.62±1.53	81.77±8.89	8.17±1.46
IB1	97.35±0.64	80.89±9.15	
IBk	89.15±0.97	81.48±7.76	
NB-gaussian	86.19±1.13	84.33±6.70	
NB-kernel	87.12±0.80	84.92±5.63	
LIBSVM	91.09±1.06	85.61±6.70	71.70±3.39
XCS	99.66±0.27	80.7±9.2	

Table D.16: Results of global comparative tests on the *ion* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	96.92±0.78	92.13±5.23	2.32±0.80
GAssist-gr2	97.19±0.51	92.44±4.56	2.16±0.38
GAssist-gr3	96.90±0.74	92.71±5.01	2.59±0.93
GAssist-inst	98.49±0.58	90.43±4.81	18.20±5.39
Majority	64.10±0.18	64.12±1.56	
C4.5	98.68±0.54	88.97±5.91	14.60±1.65
PART	98.39±0.96	90.43±5.09	7.47±1.63
IB1	100.00±0.00	87.01±4.41	
IBk	90.94±0.59	85.66±4.66	
NB-gaussian	83.44±0.89	83.23±8.06	
NB-kernel	93.00±0.42	91.50±4.70	
LIBSVM	94.19±0.64	92.14±4.62	127.77±3.22
XCS	99.86±0.24	90.1±4.7	

Table D.17: Results of global comparative tests on the *irs* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.05±0.96	94.13±6.06	3.69±0.65
GAssist-gr2	97.76±1.00	94.31±5.62	3.57±0.57
GAssist-gr3	98.33±0.79	95.20±5.87	3.91±0.58
GAssist-inst	99.47±0.41	94.49±6.34	4.19±0.91
Majority	33.33±0.00	33.33±0.00	
C4.5	98.00±0.61	94.22±5.37	4.60±0.61
PART	97.75±0.68	93.78±5.95	3.77±1.31
IB1	100.00±0.00	94.44±7.32	
IBk	96.59±0.49	94.89±6.37	
NB-gaussian	95.85±0.68	95.78±5.30	
NB-kernel	96.67±0.53	96.22±5.36	
LIBSVM	97.11±0.64	96.22±4.77	56.07±1.29
XCS	99.10±1.19	94.7±5.1	

Table D.18: Results of global comparative tests on the *lab* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	100.00±0.00	97.77±5.98	4.00±0.00
GAssist-gr2	100.00±0.00	97.75±5.79	4.00±0.00
GAssist-gr3	100.00±0.00	97.03±6.56	4.00±0.00
GAssist-inst	100.00±0.00	97.30±6.43	4.00±0.00
Majority	64.90±0.57	64.25±4.66	
C4.5	91.58±4.00	80.31±17.44	4.43±1.48
PART	94.21±2.34	80.81±16.32	3.67±0.65
IB1	99.67±1.03	86.14±15.86	
IBk	98.77±1.55	95.38±7.75	
NB-gaussian	97.66±1.16	94.68±9.97	
NB-kernel	95.92±1.60	93.76±10.50	
LIBSVM	96.04±0.93	93.35±8.32	37.67±1.62
XCS	99.92±0.24	83.5±14.8	

Table D.19: Results of global comparative tests on the *lrn* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	75.70±0.85	69.31±5.07	7.47±1.45
GAssist-gr2	75.94±0.80	69.36±5.18	7.27±1.29
GAssist-gr3	75.69±0.90	69.19±5.16	7.86±1.45
GAssist-inst	78.54±1.37	66.51±4.47	69.58±18.05
Majority	45.83±0.10	45.84±0.93	
C4.5	80.41±1.58	69.20±4.08	41.30±11.56
PART	86.73±1.08	67.49±4.30	57.53±6.32
IB1	99.69±0.13	61.62±4.99	
IBk	78.96±0.71	61.85±5.41	
NB-gaussian	73.95±0.74	71.46±4.66	
NB-kernel	75.43±0.64	71.92±4.23	
LIBSVM	73.54±0.78	66.42±3.91	428.67±3.75

Table D.20: Results of global comparative tests on the *lym* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	94.66±1.47	81.47±11.06	6.36±1.11
GAssist-gr2	94.10±1.27	79.56±11.23	6.36±1.03
GAssist-gr3	95.89±1.32	80.80±10.82	6.69±1.11
GAssist-inst	98.46±0.84	78.16±10.51	16.50±4.67
Majority	54.73±0.35	54.86±3.03	
C4.5	92.52±1.85	75.96±10.73	17.90±3.51
PART	93.34±1.83	75.87±11.44	11.10±1.62
IB1	100.00±0.00	80.58±8.10	
IBk	91.92±1.10	82.17±8.25	
NB-gaussian	87.16±1.26	83.37±8.92	
NB-kernel	88.31±1.18	83.36±10.33	
LIBSVM	93.54±1.01	84.53±9.32	96.87±2.46
XCS	98.84±0.84	79.8±10.2	

Table D.21: Results of global comparative tests on the *mmg* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	84.24±1.45	66.52±10.35	6.37±0.74
GAssist-gr2	83.77±1.35	67.50±10.39	6.25±0.56
GAssist-gr3	84.29±1.28	68.52±10.67	6.51±0.81
GAssist-inst	79.59±2.04	64.37±11.33	8.68±2.26
Majority	56.02±0.17	56.04±1.48	
C4.5	80.61±6.46	61.91±11.22	9.00±3.61
PART	75.70±5.65	62.06±11.32	4.07±1.75
IB1	100.00±0.00	62.18±10.54	
IBk	81.64±1.32	67.36±9.68	
NB-gaussian	66.19±1.38	65.20±9.71	
NB-kernel	65.78±1.15	64.86±10.01	
LIBSVM	67.15±1.13	65.46±11.43	148.00±3.40

Table D.22: Results of global comparative tests on the *pim* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	83.11±0.82	74.46±5.19	7.64±1.48
GAssist-gr2	82.63±0.82	74.32±4.75	6.98±1.26
GAssist-gr3	83.72±0.81	74.36±5.05	8.71±1.81
GAssist-inst	84.25±0.93	74.46±5.07	35.86±13.14
Majority	65.10±0.08	65.11±0.70	
C4.5	84.43±2.41	75.44±4.79	22.43±7.94
PART	78.88±1.65	74.88±4.79	6.93±1.24
IB1	100.00±0.00	71.09±4.10	
IBk	85.67±0.65	74.52±3.91	
NB-gaussian	76.42±0.65	75.17±4.46	
NB-kernel	77.07±0.61	75.30±4.45	
LIBSVM	78.27±0.53	77.32±4.70	405.90±5.92
XCS	98.90±0.67	72.4±5.3	

Table D.23: Results of global comparative tests on the *prt* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	59.42±4.24	48.29±7.74	13.97±2.32
GAssist-gr2	59.42±4.24	48.29±7.74	13.97±2.32
GAssist-gr3	59.42±4.24	48.29±7.74	13.97±2.32
GAssist-inst	59.42±4.24	48.29±7.74	13.97±2.32
Majority	24.78±0.25	24.90±2.24	
C4.5	60.41±1.66	41.45±6.13	43.20±5.07
PART	62.91±1.36	41.75±6.11	40.47±3.80
IB1	86.76±0.84	34.39±6.20	
IBk	56.47±1.39	44.12±5.84	
NB-gaussian	56.81±1.15	49.61±7.70	
NB-kernel	56.81±1.15	49.61±7.70	
LIBSVM	61.98±1.18	47.22±7.24	282.70±3.48
XCS	51.44±3.28	39.8±8.5	

Table D.24: Results of global comparative tests on the *son* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	94.66±1.46	75.79±8.64	6.86±0.98
GAssist-gr2	94.30±1.37	74.84±8.87	6.51±0.80
GAssist-gr3	97.03±1.16	77.21±8.86	7.20±1.21
GAssist-inst	96.98±1.51	78.39±8.98	18.83±6.19
Majority	53.37±0.26	53.43±2.25	
C4.5	97.97±0.51	73.21±8.61	14.27±1.63
PART	98.70±0.49	73.09±10.30	7.73±0.81
IB1	100.00±0.00	87.22±7.43	
IBk	92.01±1.38	84.63±10.14	
NB-gaussian	72.56±1.18	68.80±10.80	
NB-kernel	83.62±1.58	71.89±9.87	
LIBSVM	86.20±1.31	80.41±8.74	144.43±2.28
XCS	100.00±0.00	77.9±8.0	

Table D.25: Results of global comparative tests on the *soy* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	90.10±4.20	87.30±4.64	20.54±3.05
GAssist-gr2	90.10±4.20	87.30±4.64	20.54±3.05
GAssist-gr3	90.10±4.20	87.30±4.64	20.54±3.05
GAssist-inst	90.10±4.20	87.30±4.64	20.54±3.05
Majority	13.47±0.06	13.47±0.52	
C4.5	96.03±0.49	91.40±3.50	60.40±5.03
PART	96.07±0.62	91.22±2.67	36.90±2.40
IB1	99.74±0.15	89.75±3.16	
IBk	94.22±0.42	91.20±2.30	
NB-gaussian	93.59±0.27	92.85±2.39	
NB-kernel	93.59±0.27	92.85±2.39	
LIBSVM	95.62±0.29	93.39±2.13	543.57±6.89
XCS	78.49±4.01	85.1±4.4	

Table D.26: Results of global comparative tests on the *thy* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.85±0.55	91.92±5.18	5.34±0.63
GAssist-gr2	98.78±0.58	91.98±5.31	5.25±0.53
GAssist-gr3	98.46±0.67	92.13±5.30	5.48±0.65
GAssist-inst	99.72±0.36	95.54±3.58	5.04±0.94
Majority	69.77±0.26	69.84±2.22	
C4.5	98.47±0.54	92.68±5.87	8.20±1.01
PART	98.98±0.39	95.33±4.33	4.57±0.76
IB1	100.00±0.00	96.94±4.31	
IBk	97.57±0.65	93.97±5.20	
NB-gaussian	97.16±0.46	97.03±3.89	
NB-kernel	97.16±0.48	96.42±3.72	
LIBSVM	91.33±0.81	90.87±5.99	75.03±2.68
XCS	99.90±0.44	95.4±4.6	

Table D.27: Results of global comparative tests on the *veh* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	73.09±1.25	67.66±3.91	8.17±1.82
GAssist-gr2	73.23±1.05	68.07±4.12	7.62±1.66
GAssist-gr3	72.62±1.11	67.87±3.42	8.08±1.66
GAssist-inst	81.23±1.52	70.41±5.25	50.07±11.57
Majority	25.79±0.08	25.10±0.58	
C4.5	91.83±3.30	73.69±3.14	70.77±10.98
PART	87.75±2.63	72.71±3.60	32.63±4.29
IB1	100.00±0.00	69.70±4.20	
IBk	85.60±0.68	70.69±3.56	
NB-gaussian	46.87±0.88	45.19±3.90	
NB-kernel	64.79±0.56	61.18±4.88	
LIBSVM	73.33±0.75	71.50±3.47	674.90±2.96
XCS	98.68±0.62	74.3±4.7	

Table D.28: Results of global comparative tests on the *vot* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.93±0.32	96.81±3.12	5.75±0.77
GAssist-gr2	98.93±0.32	96.81±3.12	5.75±0.77
GAssist-gr3	98.93±0.32	96.81±3.12	5.75±0.77
GAssist-inst	98.93±0.32	96.81±3.12	5.75±0.77
Majority	61.38±0.14	61.39±1.29	
C4.5	97.18±0.27	96.30±3.64	5.73±0.51
PART	97.41±0.26	95.85±3.44	6.27±0.96
IB1	99.62±0.14	92.28±4.25	
IBk	94.34±0.45	93.04±3.92	
NB-gaussian	90.32±0.51	90.20±3.89	
NB-kernel	90.32±0.51	90.20±3.89	
LIBSVM	98.26±0.32	95.18±3.60	103.23±3.39
XCS	97.81±0.73	95.7±3.1	

Table D.29: Results of global comparative tests on the *wbcd* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	97.98±0.45	95.97±2.33	3.01±0.55
GAssist-gr2	97.95±0.38	95.88±2.36	3.09±0.61
GAssist-gr3	98.52±0.34	95.93±2.19	3.17±0.64
GAssist-inst	98.34±0.31	96.08±2.46	7.01±4.50
Majority	65.52±0.05	65.52±0.47	
C4.5	98.01±0.50	94.33±2.78	13.03±3.03
PART	98.04±0.80	94.67±3.26	10.13±2.25
IB1	99.59±0.10	95.57±2.13	
IBk	97.68±0.34	96.90±1.38	
NB-gaussian	96.08±0.33	96.05±2.40	
NB-kernel	97.57±0.18	97.52±1.76	
LIBSVM	97.22±0.26	96.95±2.20	67.93±3.13
XCS	99.37±0.39	95.9±2.3	

Table D.30: Results of global comparative tests on the *wdbc* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.22±0.40	94.18±3.27	4.72±0.85
GAssist-gr2	98.25±0.38	94.29±3.16	4.55±0.74
GAssist-gr3	98.31±0.44	93.82±2.94	4.72±0.82
GAssist-inst	99.06±0.26	96.53±2.45	6.28±3.32
Majority	62.74±0.07	62.74±0.62	
C4.5	98.87±0.44	93.32±3.28	11.17±2.08
PART	99.13±0.43	94.27±2.62	6.77±1.20
IB1	100.00±0.00	95.78±2.43	
IBk	98.47±0.28	96.83±1.84	
NB-gaussian	93.61±0.33	93.09±2.93	
NB-kernel	95.38±0.31	94.55±2.72	
LIBSVM	97.22±0.27	96.72±2.29	128.93±2.24
XCS	100.00±0.00	96.0±2.5	

Table D.31: Results of global comparative tests on the *wine* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	99.67±0.49	94.02±5.30	3.31±0.52
GAssist-gr2	99.78±0.40	93.75±4.78	3.16±0.42
GAssist-gr3	99.51±0.50	93.83±5.43	3.73±0.71
GAssist-inst	100.00±0.05	96.33±4.13	3.84±0.80
Majority	39.89±0.27	39.98±2.39	
C4.5	98.86±0.54	92.24±6.44	5.40±0.66
PART	99.15±0.47	91.71±5.62	4.47±0.50
IB1	100.00±0.00	95.62±3.82	
IBk	97.27±0.53	96.61±4.02	
NB-gaussian	98.58±0.43	96.60±4.09	
NB-kernel	98.67±0.45	97.20±3.43	
LIBSVM	99.33±0.32	98.10±3.40	73.40±2.48
XCS	100.00±0.00	95.6±4.9	

Table D.32: Results of global comparative tests on the *wpbc* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	89.72±1.28	75.06±8.88	4.13±0.47
GAssist-gr2	88.71±1.30	75.88±8.65	4.08±0.39
GAssist-gr3	91.38±1.52	74.79±8.72	4.41±0.64
GAssist-inst	89.44±1.80	76.75±8.45	7.81±3.68
Majority	76.27±0.44	76.44±3.68	
C4.5	93.66±2.70	71.11±8.58	13.20±2.24
PART	92.38±3.91	74.00±8.72	7.47±1.73
IB1	99.55±0.22	69.96±10.14	
IBk	87.15±1.14	72.95±9.10	
NB-gaussian	70.75±1.87	67.46±9.50	
NB-kernel	77.41±1.53	68.93±9.18	
LIBSVM	76.27±0.44	76.44±3.68	94.37±3.25
XCS	100.00±0.00	74.3±8.9	

Table D.33: Results of global comparative tests on the *zoo* dataset

System	Training acc.	Test acc.	Size of the solution
GAssist-gr1	98.06±1.26	91.28±8.31	7.44±0.72
GAssist-gr2	98.06±1.26	91.28±8.31	7.44±0.72
GAssist-gr3	98.06±1.26	91.28±8.31	7.44±0.72
GAssist-inst	98.06±1.26	91.28±8.31	7.44±0.72
Majority	40.60±0.57	41.10±5.04	
C4.5	98.75±0.68	92.45±6.79	10.90±2.65
PART	98.75±0.68	92.75±6.91	7.63±0.48
IB1	100.00±0.00	95.71±5.57	
IBk	96.16±0.95	94.66±5.65	
NB-gaussian	99.67±0.58	93.74±6.71	
NB-kernel	99.67±0.58	93.74±6.71	
LIBSVM	99.85±0.47	95.13±6.01	56.37±1.99
XCS	100.00±0.00	95.1±6.1	

D.2 Results on large datasets

Table D.34: Results of the global comparison tests on large datasets

Dataset	System	Training acc.	Test acc.	Size of sol.	Time (s)
adu	GAssist	85.27±0.24	85.10±0.56	10.63±2.20	300.00±0.0
	C4.5	87.53±0.15	86.05±0.38	580.67±97.43	18.28±0.26
	NB-kernel	85.30±0.06	85.21±0.51		49.93±1.97
	PART	89.37±0.11	85.68±0.52	1056.67±34.95	354.02±20.66
c-4	GAssist	71.92±2.39	71.69±2.32	21.88±8.80	300.00±0.0
	C4.5	87.50±0.31	81.02±0.44	4026.47±91.81	14.19±0.25
	NB-kernel	72.23±0.07	72.15±0.42		3.97±0.02
	PART	90.13±0.08	79.02±0.51	3716.33±42.81	1319.58±63.31
fars	GAssist	76.83±0.57	76.80±0.58	13.58±2.43	300.00±0.0
	C4.5	84.11±0.11	79.90±0.32	6216.40±276.83	55.26±0.46
	NB-kernel	79.70±0.04	79.49±0.13		14.76±0.10
	PART	86.74±0.07	78.74±0.37	4412.90±37.76	3806.33±98.42
hyp	GAssist	94.77±0.50	94.47±0.77	6.85±0.85	300.00±0.0
	C4.5	99.81±0.04	99.57±0.27	14.60±0.88	1.61±0.47
	NB-kernel	96.35±0.22	95.97±0.71		1.38±0.25
	PART	99.83±0.04	99.51±0.28	10.30±1.46	2.02±0.35
krkp	GAssist	97.74±1.14	97.58±1.32	7.46±0.58	300.00±0.0
	C4.5	99.64±0.07	99.43±0.40	29.03±2.07	1.47±0.17
	NB-kernel	88.16±0.28	87.84±1.91		1.42±0.19
	PART	99.72±0.06	99.06±0.65	22.17±2.56	1.76±0.02
mush	GAssist	99.96±0.14	99.95±0.19	4.96±0.78	300.00±0.0
	C4.5	100.00±0.00	100.00±0.00	25.00±0.00	1.64±0.38
	NB-kernel	95.82±0.09	95.79±0.74		1.30±0.40
	PART	100.00±0.00	100.00±0.00	11.67±1.64	1.82±0.36
nur	GAssist	95.39±0.93	95.23±1.10	19.80±7.21	300.00±0.0
	C4.5	98.16±0.06	97.18±0.47	356.63±10.04	1.73±0.48
	NB-kernel	90.37±0.12	90.28±0.67		1.37±0.37
	PART	99.76±0.05	99.14±0.42	196.63±11.18	2.54±0.08
pen	GAssist	72.18±2.68	71.93±2.96	12.68±1.75	300.00±0.0
	C4.5	99.27±0.06	96.50±0.44	188.17±5.98	3.71±0.55
	NB-kernel	89.09±0.14	88.49±1.07		10.07±0.28
	PART	99.56±0.06	96.92±0.60	79.27±4.68	7.15±0.54
sat	GAssist	80.45±0.62	80.00±1.30	8.25±1.56	300.00±0.0
	C4.5	97.57±0.25	86.07±1.46	278.83±11.38	4.30±0.57
	NB-kernel	82.56±0.19	82.06±1.64		6.64±0.52
	PART	98.39±0.40	86.54±1.53	161.37±8.71	15.11±1.13
seg	GAssist	90.89±1.08	90.25±2.11	8.09±1.06	300.00±0.0
	C4.5	99.19±0.14	96.54±1.58	41.10±2.41	1.24±0.24
	NB-kernel	86.52±0.32	85.82±1.95		4.55±0.58
	PART	99.47±0.21	96.78±1.26	27.87±2.20	2.10±0.23
sick	GAssist	98.75±0.23	98.41±0.70	6.33±0.66	300.00±0.0
	C4.5	99.63±0.14	98.67±0.51	29.03±4.25	1.80±0.41
	NB-kernel	96.02±0.16	95.82±0.95		1.29±0.25
	PART	99.63±0.12	98.65±0.54	18.73±2.54	1.84±0.46
spl	GAssist	93.58±2.53	92.46±2.79	11.67±4.23	300.00±0.0
	C4.5	96.24±0.22	94.11±1.05	172.93±10.91	1.27±0.36
	NB-kernel	95.89±0.18	95.36±1.14		0.97±0.29
	PART	97.31±0.26	92.46±1.19	101.60±7.65	2.79±0.36
wav	GAssist	78.28±0.60	76.01±1.97	10.64±1.94	300.00±0.0
	C4.5	97.29±0.61	75.93±2.10	294.60±12.97	4.07±0.48
	NB-kernel	81.59±0.21	79.89±1.40		17.98±0.28
	PART	92.29±1.65	78.07±1.82	86.60±10.08	14.81±1.14

APPENDIX D. FULL RESULTS OF THE GLOBAL COMPARISON WITH ALTERNATIVE MACHINE LEARNING SYSTEMS

Bibliography

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, *7*(1), 39–59.

Aguilar, J., Bacardit, J., & Divina, F. (2004). Experimental evaluation of discretization schemes for rule induction. In *GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 828–839. Springer-Verlag, LNCS 3102.

Aguilar-Ruiz, J., Riquelme, J., & Toro, M. (2003, April). Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, *33*(2), 324–331.

Aguilar-Ruiz, J. S., Riquelme, J., & Toro, M. (2000). Data set editing by ordered projection. In *Proceedings of the European Conference on Artificial Intelligence* pp. 251–255. IOS Press.

Aguirre, E., González, A., & Pérez, R. (2002). Un modelo de selección de características para los algoritmos de aprendizaje basados en el modelo pittsburgh. In *"Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)"* pp. 354–360.

Aha, D. W., Kibler, D. F., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Bacardit, J., & Butz, M. V. (2004). Data mining in learning classifier systems: Comparing xcs with gassist. In *Proceedings of the 7th International Workshop on Learning Classifier Systems* (in press), LNAI, Springer-Verlag.

Bacardit, J., & Garrell, J. M. (2002a). Evolution of adaptive discretization intervals for A rule-based genetic learning system. In *Proceedings of the Fourth Genetic and Evolutionary Computation Conference* pp. 677. Morgan Kaufmann Publishers.

Bacardit, J., & Garrell, J. M. (2002b). Evolution of multi-adaptive discretization intervals for A rule-based genetic learning system. In *Proceedings of the Eighth Iberoamerican Conference on Artificial Intelligence* pp. 350–360. LNAI vol. 2527, Springer-Verlag.

Bacardit, J., & Garrell, J. M. (2002c). Métodos de generalización para sistemas clasificadores de Pittsburgh. In *"Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)"* pp. 486–493.

Bacardit, J., & Garrell, J. M. (2002d). The role of interval initialization in a gbml system with rule representation and adaptive discrete intervals. In *Proceedings of the Fifth Catalan Conference on Artificial Intelligence* pp. 184–195. LNAI vol. 2504, Springer-Verlag.

Bacardit, J., & Garrell, J. M. (2003a). Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In *Proceedings of the 6th International Workshop on Learning Classifier Systems* (in press), LNAI, Springer-Verlag.

Bacardit, J., & Garrell, J. M. (2003b). Comparison of training set reduction techniques for pittsburgh approach genetic classifier systems. In *"X Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA2003)"*, Volume 1 pp. 223–226.

Bacardit, J., & Garrell, J. M. (2003c). Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003* pp. 1818–1831. LNCS 2724, Springer-Verlag.

Bacardit, J., & Garrell, J. M. (2003d). Incremental learning for pittsburgh approach classifier systems. In *"Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados."* pp. 303–311.

Bacardit, J., & Garrell, J. M. (2004). Analysis and improvements of the adaptive discretization intervals knowledge representation. In *GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 726–738. Springer-Verlag, LNCS 3103.

Bacardit, J., Goldberg, D., Butz, M., Llorá, X., & Garrell, J. M. (2004). Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy. In *Parallel Problem Solving from Nature - PPSN 2004* pp. 1021–1031. Springer-Verlag, LNCS 3242.

Bacardit, J., Goldberg, D. E., & Butz, M. V. (2004). Improving the performance of a pittsburgh learning classifier system using a default rule. In *Proceedings of the 7th International Workshop on Learning Classifier Systems* (in press), LNAI, Springer-Verlag.

Bäck, T. (1995). Generalized convergence models for tournament- and (mu, lambda)-selection. In *Proceedings of the 6th International Conference on Genetic Algorithms* pp. 2–8. Morgan Kaufmann Publishers Inc.

Bassett, J. K., & Jong, K. A. D. (2000). Evolving behaviors for cooperating agents. In *International Syposium on Methodologies for Intelligent Systems* pp. 157–165. Springer-Verlag.

Bernadó, E., & Garrell, J. M. (2000). Multiobjective learning in a genetic classifier system (MOLeCS). *Butlletí de l'Associació Catalana l'Intel.ligència Artificial*, *22*, 102–111.

Bernadó, E., Mekaouche, A., & Garrell, J. M. (1999). A Study of a Genetic Classifier System Based on the Pittsburgh Approach on a Medical Domain. In *12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-99* pp. 175–184. Springer-Verlag.

Blake, C., Keogh, E., & Merz, C. (1998). UCI repository of machine learning databases. (www.ics.uci.edu/mlearn/MLRepository.html).

Booker, L. B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment*. Doctoral dissertation, The University of Michigan.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks.

Brodley, C. (1993). Addressing the selective superiority problem: Automatic algorithm /model class selection. In *Proceedings of the Tenth International Conference on Machine Learning* pp. 17–24. Morgan Kaufmann Publishers.

Burke, D. S., Jong, K. A. D., Grefenstette, J. J., Ramsey, C. L., & Wu, A. S. (1998, Winter). Putting more genetics into genetic algorithms. *Evolutionary Computation*, *6*(4), 387–410.

Butz, M. V. (2004). *Rue-based evolutionary online learning systems: Learning bounds, classification and prediction*. Doctoral dissertation, University of Illinois at Urbana-Champaign.

Butz, M. V., Sastry, K., & Goldberg, D. E. (2003). Tournament selection in XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003* pp. 1857–1869. LNCS 2724, Springer-Verlag.

Cantú-Paz, E. (2002). On random numbers and the performance of genetic algorithms. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 311–318. Morgan Kaufmann Publishers.

Cantu-Paz, E., & Kamath, C. (2003, Feb). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *7*(1), 54–68.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *Proceedings of the European working session on learning on Machine learning* pp. 164–178. Springer-Verlag New York, Inc.

Cerquides, J., & de Mantaras, R. L. (1997). Proposal and empirical comparison of a parallelizable distance-based discretization method. In *III International Conference on Knowledge Discovery and Data Mining* pp. 139–142. AAAI Press.

Chan, C.-C., Batur, C., & Srinivasan, A. (1991). Determination of quantization intervals in rule based model for dynamic systems. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics* pp. 1719– 1723. IEEE Press.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Department of Computer Science and Information Engineering, National Taiwan University. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proc. Fifth European Working Session on Learning* pp. 151–163. Berlin: Springer-Verlag.

Clark, P., & Niblett, T. (1989). The cn2 induction algorithm. *Machine Learning*, *3*, 261–283.

Cohen, W. W. (1995, July 9–12,). Fast effective rule induction. In Prieditis, A., & Russell, S. (Eds.), *Proc. of the 12th International Conference on Machine Learning* pp. 115–123. Tahoe City, CA: Morgan Kaufmann.

Corcoran, A. L., & Sen, S. (1994). Using real-valued genetic algorithms to evolve rule sets for classification. In *Proceedings of the IEEE Conference on Evolutionary Computation* pp. 120–124. IEEE Press.

Cordón, O., Herrera, F., Hoffmann, F., & Magdalena, L. (2001). *Genetic fuzzy systems. evolutionary tuning and learning of fuzzy knowledge bases*. World Scientific.

Cordon, O., Herrera, F., & Villar, P. (2001, Aug). Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on Fuzzy Systems*, *9*(4), 667–674.

Cottrell, G. W. (1990). Extracting features from faces using compression networks: Face, identity, emotion, and gender recognition using holons. In Touretzky, D. S., Elman, J. L., Sejnowski, T. J., & Hinton, G. E. (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School* pp. 328–337. Morgan Kaufmann.

Dasgupta, D., & Gonzalez, F. A. (2001). Evolving complex fuzzy classifier rules using a linear tree genetic representation. In *Proceedings of the Third Genetic and Evolutionary Computation Conference* pp. 299–305. Morgan Kaufmann.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Mach. Learn.*, *3*(2-3), 121–138.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, *6*(1), 81–92.

DeJong, K. A., & Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence* pp. 651–656. Morgan Kaufmann.

DeJong, K. A., Spears, W. M., & Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, *13*(2/3), 161–188.

Divina, F., Keijzer, M., & Marchiori, E. (2003, 12-16 July). A method for handling numerical attributes in GA-based inductive concept learners. In *GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 898–908. Springer-Verlag.

Domingos, P. (1994). The rise system: Conquering without separating. In *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence* pp. 704–707. IEEE Press.

Ekart, A., & Nemeth, S. Z. (2001). Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, *2*(1), 61–73.

Elomaa, T., & Rousu, J. (2002). Fast minimum training error discretization. In *Proceedings of the Nineteenth International Conference on Machine Learning* pp. 131–138. Morgan Kaufmann Publishers Inc.

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* pp. 1022–1029. Morgan Kaufmann.

Fogel, L. J. (1964). *On the organization of intellect*. Doctoral dissertation, University of California, Los Angeles.

Forsyth, R., Clarke, D. D., & Wright, R. L. (1994). Overfitting revisited: An information-theoretic approach to simplifying discrimination trees. *Journal of Experimental and Theoretical Artificial Intelligence*, *6*(3), 289–302.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proc. 15th International Conf. on Machine Learning* pp. 144–151. Morgan Kaufmann, San Francisco, CA.

Freitas, A. A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer-Verlag.

Fürnkranz, J. (1998). Integrative windowing. *Journal of Artificial Intelligence Research*, *8*, 129–164.

Fürnkranz, J. (1999, February). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*(1), 3–54.

Gao, Q., Li, M., & Vitányi, P. (2000). Applying mdl to learn best model granularity. *Artificial Intelligence*, *121*(1-2), 1–29.

Gathercole, C., & Ross, P. (1994, 9-14). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature III*, Volume 866 pp. 312–321. Jerusalem: Springer-Verlag.

Giráldez, R., Aguilar-Ruiz, J., & Riquelme, J. (2003, 12-16 July). Natural coding: A more efficient representation for evolutionary learning. In *GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 979–990. Springer-Verlag.

Giráldez, R., Aguilar-Ruiz, J. S., Riquelme, J. C., Ferrer, J., & Rodríguez, D. (2002). Discretization oriented to decision rule generation. In *Proceedings of the International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES'02* pp. 275–279. IOS Press.

Giraud-Carrier, C. (2000). A note on the utility of incremental learning. *AI Commun.*, *13*(4), 215–223.

Goldberg, D. E. (1989a). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Publishing Company, Inc.

Goldberg, D. E. (1989b). Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)* pp. 70–79. Morgan Kaufmann.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer Academic Publishers.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms* pp. 69–93. Morgan Kaufmann.

Goldberg, D. E., Sastry, K., & Latoza, T. (2001, 7-11). On the supply of building blocks. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., & Burke, E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* pp. 336–342. San Francisco, California, USA: Morgan Kaufmann.

Golobardes, E., Llorà, X., Garrell, J. M., Vernet, D., & Bacardit, J. (2000). Genetic classifier system as a heuristic weighting method for a case-based classifier system. *Butlletí de l'Associació Catalana d'Intel.ligència Artificial*, *22*, 132–141.

Goulden, C. (1956). *Methods of statistical analysis*. New York: John Wiley & Sons.

Grefenstette, J. J. (1991). Lamarckian learning in multi-agent environments. In Belew, R., & Booker, L. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* pp. 303–310. San Mateo, CA: Morgan Kaufman.

Harik, G. (1995). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor. Also available as IlliGAL Report 97005.

Harik, G., Cantu-Paz, E., Goldberg, D. E., & Miller, B. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the IEEE International Conference on Evolutionary Computation* pp. 7–12. IEEE Press.

Ho, K. M., & Scott, P. D. (1997). Zeta: A global method for discretization of continuous variables. In *III International Conference on Knowledge Discovery and Data Mining* pp. 191–194. AAAI Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Hayes-Roth, D., & Waterman, F. (Eds.), *Pattern-directed Inference Systems* (pp. 313–329). New York: Academic Press.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–91.

Iba, H., de Garis, H., & Sato, T. (1994). Genetic programming using a minimum description length principle. In Kinnear, Jr., K. E. (Ed.), *Advances in Genetic Programming* (pp. 265–284). MIT Press.

Janikow, C. (1991). *Indictive learning of decision rules in attribute-based examples: a knowledge-intensive genetic algorithm approach*. Doctoral dissertation, University of North Carolina.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Mach. Learn.*, *13*(2-3), 189–228.

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, *6*(5), 429–450.

Joachims, T. (2002). *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers.

John, G. H., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* pp. 338–345. Morgan Kaufmann Publishers, San Mateo.

John, G. H., & Langley, P. (1996, 2–4). Static versus dynamic sampling for data mining. In Simoudis, E., Han, J., & Fayyad, U. M. (Eds.), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* pp. 367–370. AAAI Press.

Kavcic, A., & Srinivasan, M. (2001). The minimum description length principle for modeling recording channels. *IEEE Journal on Selected Areas in Communications*, *19*(4), 719–729.

Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence* pp. 123–128. San Jose, CA: AAAI Press.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence* pp. 1137–1145. Morgan Kaufmann.

Koza, J. R. (1992). *Genetic programming*. Cambridge, Massachusetts: The MIT Press.

Kozlov, A. V., & Koller, D. (1997). Nonuniform dynamic discretization in hybrid networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in AI (UAI)* pp. 314–325. Morgan Kaufmann.

Lang, K. J., Hinton, G. E., & Waibel, A. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks*, *3*(1), 23–43.

Langdon, W. B. (1997, 14 May). *Fitness causes bloat in variable size representations* (Technical Report CSRP-97-14). University of Birmingham, School of Computer Science. Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97.

Langley, P. (1995). *Elements of machine learning*. Morgan Kaufmann Publishers Inc.

Langley, P., Iba, W., & Thompson, K. (1992). An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence* pp. 223–228. AAAI Press.

Larranaga, P., & Lozano, J. (Eds.) (2002). *Estimation of Distribution Algorithms, A New Tool for Evolutionnary Computation*. Genetic Algorithms and Evolutionnary Computation. Kluwer Academic Publishers.

Lavrac, N., & Dzeroski, S. (1993). *Inductive logic programming: Techniques and applications*. Routledge.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989, Winter). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Lehtokangas, M., Saarinen, J., Huuhtanen, P., & Kaski, K. (1996). Predictive minimum description length criterion for time series modeling with neural networks. *Neural Computation*, *8*(3), 583–593.

Liu, H., Hussain, F., Tam, C. L., & Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, *6*(4), 393–423.

Liu, H., & Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of Seventh IEEE International Conference on Tools with Artificial Intelligence* pp. 388–391. IEEE Computer Society.

Llorà, X., & Garrell, J. M. (2001a). Inducing partially-defined instances with evolutionary algorithms. In *Proceedings of the Eighteenth International Conference on Machine Learning* pp. 337–344. Morgan Kaufmann.

Llorà, X., & Garrell, J. M. (2001b). Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In *Proceedings of the Third Genetic and Evolutionary Computation Conference* pp. 461–468. Morgan Kaufmann.

Llorà, X., Goldberg, D. E., Traus, I., & Bernadó, E. (2002). Accuracy, Parsimony, and Generality in Evolutionary Learning System a Multiobjective Selection. In *Advances in Learning Classifier Systems: proceedings of the 5th International Workshop on Learning Classifier Systems* (in press), LNAI, Springer-Verlag.

Llorá, X., & Wilson, S. (2004). Mixed decision trees: Minimizing knowledge representation bias in lcs. In *GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 797–809. Springer-Verlag, LNCS 3103.

Luke, S., & Panait, L. (2002). Lexicographic parsimony pressure. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* pp. 829–836. Morgan Kaufmann.

Maloof, M. A., & Michalski, R. S. (2000). Selecting examples for partial memory learning. *Machine Learning*, *41*(1), 27–52.

Martí, J., Cufí, X., Regincós, J., & et al. (1998). Shape-based feature selection for microcalcification evaluation. In *Proceedings of the SPIE Medical Imaging Conference on Image Processing* pp. 1215–1224. SPIE Press.

Martínez Marroquín, E., Vos, C., & et al. (1996). Morphological analysis of mammary biopsy images. In *Proceedings of the IEEE International Conference on Image Processing* pp. 943–947. IEEE Press.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *8*(1), 3–30.

McCulloch, W., & Pitts, W. (1943). A logical calculus of ideas immanent in neural activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*. Springer-Verlag.

Michalski, R. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, Volume A3 pp. 125–128.

Michalski, R. S., Mozetic, I., & Hong, J. (1986). The multi-purpose incremental learning system aq15 and it testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence* pp. 1041–1045. AAAI Press.

Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986, July). *The AQ15 inductive learning system: an overview and experiments* (Technical Report UIUCDCS-R-86-1260). University of Illinois.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann Publishers Inc.

Nordin, P., & Banzhaf, W. (1995, 15-19). Complexity compression and evolution. In Eshelman, L. (Ed.), *Genetic Algorithms: Proceedings of the Sixth International Conference* pp. 310–317. Pittsburgh, PA, USA: Morgan Kaufmann.

Oei, C. K., Goldberg, D. E., & Chang, S.-J. (1991). *Tournament selection, niching, and the preservation of diversity* (IlliGAL Report No. 91011). Urbana, IL: University of Illinois at Urbana-Champaign.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers Inc.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999, 13-17). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I pp. 525–532. Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Pfahringer, B. (1994). Controlling constructive induction in CIPF: An MDL approach. In *Proc. of the European Conference on Machine Learning ECML-94*, Volume 784 of *LNAI* pp. 242–256. Springer-Verlag.

Pfahringer, B. (1995). *Practical uses of the minimum description length principle in inductive learning*. Doctoral dissertation, Institut fur Med.Kybernetik u. AI, Technische Universitat Wien.

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning* (pp. 185–208). MIT Press.

Pomerleau, D. (1993). *Neural network perception for mobile robot guidance*. Kluwer Academic Publishing.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81 – 106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Quinlan, J. R. (1995). Mdl and categorical theories (continued). In *Proceedings of the 12th International Conference on Machine Learning* pp. 464–470. Morgan Kaufmann.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, *80*(3), 227–248.

Radcliffe, N. J. (1990). *Genetic neural networks on mimd computers*. Doctoral dissertation, University of Edinburgh.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *vol. 14*, 465–471.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, *2*(3), 229–246.

Roth, P. (1994). Missing data: A conceptual review for applied psychologists. *Personnel Psychology*, *47*(3), 537–560.

Rudolph, G. (1998). Finite Markov Chain Results in Evolutionary Computation: A Tour d'Horizon. *Fundamenta Informaticae*, *35*(1–4), 67–89.

Russel, S., & Norvig, P. (1995). *Artificial intelligence, a modern approach*. Englewood Cliffs, NJ: Prentice Hall.

Salamó, M., & Golobardes, E. (2002). Deleting and Building Sort Out Techniques for Case Base Maintenance. In *Advances in Case-Based Reasoning. 6th. European Conference on Case-Based Reasoning* pp. 365–379. Springer-Verlag.

Salamó, M., & Golobardes, E. (2003). Hybrid deletion policies for case base maintenance. In *Proceedings of FLAIRS-2003* pp. 150–154. AAAI Press.

Shaffer, J. P. (1995). Multiple hypothesis testing. *Annu. Rev. Psychol.*, *46*, 561–584.

Shannon, C. E., & Weaver, W. (1949). *The mathematical theory of communication*. Urbana: University of Illinois Press.

Sharpe, P. K., & Glover, R. P. (1999). Efficient ga based techniques for classification. *Applied Intelligence*, *11*(3), 277–284.

Sierra, B., Lazkano, E., Inza, I., Merino, M., Larrañaga, P., & Quiroga, J. (2001). Prototype selection and feature subset selection by estimation of distribution algorithms. A case study in the survival of cirrhotic patients treated with TIPS. *Lecture Notes in Computer Science*, *2101*, 20–29.

Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *International Conference on Machine Learning* pp. 293–301. Morgan Kaufmann.

Smith, S. (1980). *A learning system based on genetic algorithms*. Doctoral dissertation, University of Pittsburgh.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* pp. 421–425. Los Altos, CA: Morgan Kaufmann.

Soule, T., & Foster, J. A. (1998, Winter). Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, *6*(4), 293–309.

Stone, C., & Bull, L. (2003). For real! xcs with continuous-valued inputs. *Evolutionary Computation Journal*, *11*(3), 298–336.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms* pp. 2–9. Morgan Kaufmann Publishers Inc.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S. fourth edition*. Springer-Verlag. ISBN 0-387-95457-0.

Venturini, G. (1993). Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts. In Brazdil, P. B. (Ed.), *Machine Learning: ECML-93 - Proc. of the European Conference on Machine Learning* (pp. 280–296). Berlin, Heidelberg: Springer-Verlag.

Vose, M. D. (1999). *The simple genetic algorithm : Foundations and theory*. Complex Adaptative Systems. Bradford Books.

Wang, K., & Liu, B. (1998). Concurrent discretization of multiple attributes. In *Proceedings of the 5th Pacific Rim International Conference on Topics in Artificial Intelligence (PRICAI-98)* pp. 250–259. LNAI 1531, Springer-Verlag.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, *38*(3), 257–286.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (1999). Get real! XCS with continuous-valued inputs. In Booker, L., Forrest, S., Mitchell, M., & Riolo, R. L. (Eds.), *Festschrift in Honor of John H. Holland* pp. 111–121. Center for the Study of Complex Systems.

Wilson, S. W. (2002). Compact rulesets from xcsi. In *Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems* pp. 197–210. Springer-Verlag.

Witten, I. H., & Frank, E. (2000). *Data Mining: practical machine learning tools and techniques with java implementations*. Morgan Kaufmann.

Wolberg, W., & Mangasarian, O. (1990). Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology. *PNAS*, *87*(23), 9193–9196.

Wolpert, D. H., & Macready, W. G. (1995, February). *No free lunch theorems for search* (Working Papers 95-02-010). Santa Fe Institute. available at http://ideas.repec.org/p/wop/safiwp/95-02-010.html.

Wong, M. L., Lam, W., & Leung, K. S. (1999). Using evolutionary programming and minimum description length principle for data mining of bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(2), 174–178.

Yang, Y., & Webb, G. I. (2002). Non-disjoint discretization for naive-bayes classifiers. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML '02)* pp. 666– 673. San Francisco: Morgan Kaufmann.

Zadeh, L. (1965). Information and control. L. Zadeh. Fuzzy Sets. Information and Control, 8:338–353, 1965.