Hybrid Genetic Relational Search for Inductive Learning

Federico Divina

This research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 612-052-001.

SIKS Dissertation Series No. 2004-16. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

VRIJE UNIVERSITEIT

Hybrid Genetic Relational Search for Inductive Learning

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op dinsdag 26 oktober 2004 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Federico Divina

geboren te Borgo Valsugana, Italië

promotor:	prof.dr. A.E. Eiben
copromotor:	dr. E. Marchiori

Contents

Chapter 1

Introduction

1.1 Inductive Concept Learning

An important characteristic of all natural systems is the ability to acquire knowledge through experience and to adapt to new situations. Learning is the single unifying theme of all natural systems. One of the basic ways of gaining knowledge is through examples of some concepts. For instance, we may learn how to distinguish a dog from other creatures after that we have seen a number of creatures, and after that someone (a teacher, or supervisor) told us which creatures are dogs and which are not. This way of learning is called supervised learning.

Inductive Concept Learning (ICL) (Mitchell, 1982) constitutes a central topic in machine learning. The problem can be formulated in the following manner: given a description language used to express possible hypotheses, a background knowledge, a set of positive examples, and a set of negative examples, one has to find a hypothesis which covers all positive examples and none of the negative ones (cf. (Kubat et al., 1998; Mitchell, 1997)). This is a supervised way of learning, since a supervisor has already classified the examples of the concept into positive and negative examples. The so learned concept can be used to classify previously unseen examples.

In general deriving general conclusions from specific observation is called induction. Thus in ICL, concepts are induced because obtained from the observation of a limited set of training examples. The process can be seen as a search process (Mitchell, 1982). Starting from an initial hypothesis, what is done is searching the space of the possible hypotheses for one that fits the given set of examples.

A representation language has to be chosen in order to represent concepts, examples and the background knowledge. This is an important choice, because this may limit the kind of concept we can learn. With a representation language that has a low expressive power we may not be able to represent some problem domain, because too complex for the language adopted. On the other side, a

too expressive language may give us the possibility to represent all problem domains. However this solution may also give us too much freedom, in the sense that we can build concepts in too many different ways, and this could lead to the impossibility of finding the right concept.

1.2 Motivations

We are interested in learning concepts expressed in a fragment of first–order logic (FOL). This subject is known as Inductive Logic Programming (ILP), where the knowledge to be learn is expressed by Horn clauses, which are used in programming languages based on logic programming like Prolog.

Learning systems that use a representation based on first–order logic have been successfully applied to relevant real life problems, e.g., learning a specific property related to carcinogenicity.

Learning first–order hypotheses is a hard task, due to the huge search space one has to deal with. The approach used by the majority of ILP systems tries to overcome this problem by using specific search strategies, like the top-down and the inverse resolution mechanism (see chapter 2). However, the greedy selection strategies adopted for reducing the computational effort, render techniques based on this approach often incapable of escaping from local optima.

An alternative approach is offered by genetic algorithms (GAs). GAs have proved to be successful in solving comparatively hard optimization problems, as well as problems like ICL. GAs represents a good approach when the problems to solve are characterized by a high number of variables, when there is interaction among variables, when there are mixed types of variables, e.g., numerical and nominal, and when the search space presents many local optima. Moreover it is easy to hybridize GAs with other techniques that are known to be good for solving some classes of problems.

Another appealing feature of GAs is represented by their intrinsic parallelism, and their use of exploration operators, which give them the possibility of escaping from local optima. However this latter characteristic of GAs is also responsible for their rather poor performance on learning tasks which are easy to tackle by algorithms that use specific search strategies.

These observations suggest that the two approaches above described, i.e., standard ILP strategies and GAs, are applicable to partly complementary classes of learning problems. More important, they indicate that a system incorporating features from both approaches could profit from the different benefits of the approaches.

This motivates the aim of this thesis, which is to develop a system based on GAs for ILP that incorporates search strategies used in successful ILP systems. Our approach is inspired by memetic algorithms (Moscato, 1989), a population based search method for combinatorial optimization problems. In evolutionary computation memetic algorithms are GAs in which individuals can be refined during their lifetime.

1.3 Objectives of the Thesis

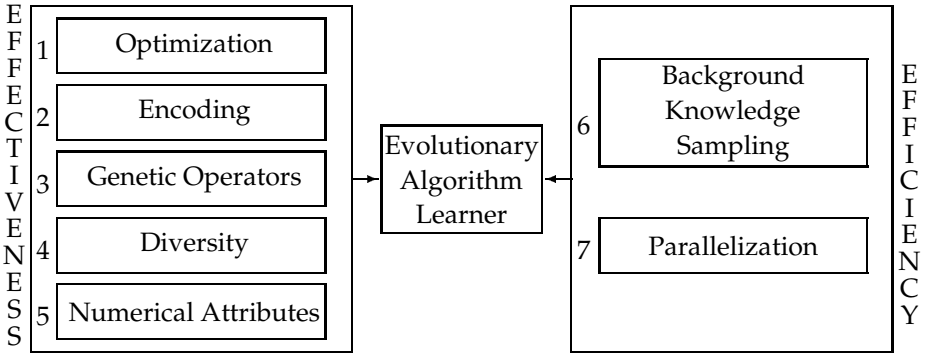


Figure 1.1: Components of an evolutionary learning system.

As already stated in the previous section, the aim of this thesis is to design a ILP system that incorporates standard ILP strategies and GAs techniques. The issues of efficiency and effectiveness are central in our research. These issues are addressed along the dimensions illustrated in figure 1.1, and briefly explained in the sequel.

The main features of the system for achieving effectiveness and efficiency are illustrated in figure 1.1 and explained in the following. First we will address the features regarding the effectiveness of the system and then those regarding the efficiency.

Effectiveness

1. Incorporate into a GA an **optimization** phase based on ILP operators for optimizing individuals of the current population. This helps to guide the GA search towards regions of the search space containing good individuals (exploitation).
2. Develop a **representation** close to the Prolog syntax. This choice is motivated by the fact that such a representation makes the application of relational operators used in ILP and the evaluation of individuals easier.
3. Develop **genetic operators** that bias the search toward better hypotheses. Standard GA operators act *blindly*, that is they do not incorporate knowledge information about the problem. The operators we introduce act *greedily*. They take into consideration various possibilities, and choose the one yielding the best improvement in terms of fitness.

4. Develop techniques for promoting **diversity** in the population and good coverage of positive examples. When learning concepts with a GA based system, it is fundamental that the population be diverse and that as many examples as possible be covered. We want to assure that these two aspects are met by the population evolved by our system.
5. Introduce methods for handling **numerical attributes**. Many learning problems use data containing numerical attributes. Numerical attributes affect the efficiency of the learning process and the accuracy of the learned theory.

Efficiency

6. Develop techniques for reducing the computational cost (in terms of time) of the learning process. More precisely, employing mechanisms for controlling the computational cost of fitness evaluation and the computational cost of the genetic search.
7. Exploit the natural **parallelism** of the GAs. We want to parallelize the system in order to reduce to the computational effort for carrying out the process.

Points 1,2,3,4 and 6 are discussed in chapter 5. We address point 5 in chapter 6, while chapter 8 regards point 7.

1.4 Overview of the Thesis

The thesis is structured in the following way. In chapter 2 we give a brief introduction to inductive logic programming. We begin by explaining the limits of a propositional representation, used for representing concepts, examples and background knowledge, and why in some cases a first–order representation is needed. We then present the basic concepts of inductive logic programming. We end the chapter by giving two examples of standard algorithms for inductive logic programming.

In chapter 3, the basic notions of evolutionary computation are given. We begin by individuating four paradigms in which evolutionary computation can be divided, and by giving some history of the field. We then discuss various aspects of evolutionary computation when applied to ICL. The reasons for which the entire hypothesis space can not be considered and some method for limiting the portion of the hypothesis space to consider are then presented. The concepts of diversity, species and niches are then given. The chapter ends with an explanation on how evolutionary computation and other heuristics can be combined in order to obtain better results.

Chapter 4 gives an overview of state of the art ILP algorithms based on evolutionary computation. Five systems are briefly presented. The first three

presented systems adopt the same encoding, while the other two adopt different solutions for representing candidate solutions. A description of all the genetic operators that are used by the systems is given.

In chapter 5 we describe in detail the system subject of this thesis. All the basic features are presented and discussed in this chapter. We start by giving a general explanation of the system. Then we present one by one all the components, starting from the particular biases adopted for limiting the search space, and ending with the way in which a final solution is extracted.

The way in which the system handles numerical values is the subject of chapter 6. We propose three alternative ways in which this can be done. In this chapter a standard way for dealing with numerical values is also briefly introduced.

A number of experiments for testing the various components of the introduced system and for comparing the performance of the system with other systems are presented in chapter 7. With these experiments we have evaluated the effectiveness of some solutions adopted by the system.

In chapter 8 a simple parallelization of the system is described, and some experiments are conducted for evaluating the effectiveness of the parallelization.

Two case studies are presented in chapter 9. The first study does not require a first–order representation. This case regards the analysis of a medical dataset. Two problems are extracted from this dataset. The first problem is to extract rules for individuating whether a patient is satisfied by his or her relation with his or her doctor. The second problem consists in extracting rules for individuating psychiatric patients and non–psychiatric patients. If such rules can be found, this may mean that in psychiatry the doctor–patient relation is perceived in a different way. The second case study requires a first–order representation, and regards the acquisition of knowledge for individuating traffic problems on a road network.

Finally, in chapter 10 we give some conclusions for this thesis.

1.5 Notation

The following notation is adopted in this thesis:

E denotes a set of examples;

e denotes a single example;

E^+ denotes a set of positive examples;

E^- denotes a set of negative examples;

φ denotes an individual of the population;

C denotes a clause;

$\square$ denotes the empty clause;

L denotes a literal;

P denotes a predicate;

t denotes a term;

$X, Y, Z, \ldots$ denote variables. We follow Prolog notation, so in general terms starting with a capital letter denote variables, while terms starting with a lower case letter denote constants;

θ denotes a substitution;

$\leftarrow L_1 \ldots L_n$ denotes a query;

H denotes an hypothesis;

p_x denotes the set of positive examples covered by x, where x can be either φ, H or C;

n_x denotes the set of negative examples covered by x, where x can be either φ, H or C;

$Cov(e)$ denotes the set of individuals covering the example e;

$Cov_\varphi = [p_\varphi / n_\varphi]$ denotes the number p_φ of positive and n_φ of negative examples covered by an individual φ;

BK denotes the background knowledge;

$f(x)$ denotes the fitness of x, where x can be either H, C or φ;

Chapter 2

Inductive Logic Programming

Learning from examples in FOL, also known as Inductive Logic Programming (ILP) (Muggleton and Raedt, 1994), constitutes a central topic in Machine Learning, with relevant applications to problems in complex domains like natural language and molecular computational biology (Muggleton, 1999).

Learning can be viewed as a search problem in the space of all possible hypotheses (Mitchell, 1982). Given a FOL description language used to express possible hypotheses, a background knowledge, a set of positive examples, and a set of negative examples, one has to find a hypothesis which covers all positive examples and none of the negative ones (cf. (Kubat et al., 1998; Mitchell, 1997)).

This problem is NP-hard even if the language to represent hypotheses is propositional logic. When FOL hypotheses are used, the complexity of searching is combined with the complexity of evaluating hypotheses (Giordana and Saitta, 2000).

In this chapter we give a brief introduction to ILP. In section 2.1, we start by motivating the choice of first–order logic for describing a learning problem.

Section 2.2 gives an introduction to ILP. In particular we first give some basic definitions of FOL, which are needed in the following of this thesis. We then address the problem of verifying if a given hypothesis covers an example. The way in which the space of all possible hypotheses can be ordered is then discussed. The ordering presented is exploited by many systems for ILP. In section 2.3 we give two examples of systems that can solve ILP problems. Section 2.4 concludes this chapter by summarizing the treated arguments.

2.1 Representation Language

When we want to solve a problem with a computer, the first thing that should be done is to translate the problem into computational terms. In our case this means to choose a representation language and an encoding.

The choice of a representation language for representing hypotheses may vary from a fragment of propositional calculus to second–order logic. While the former has a low expressive power, the latter is rather complex, and for this reason is seldom used. In this chapter we only address the representation language issue, while we will address the encoding issue in chapter 3.

Car	Price	Conditions	Age	Power (cc)	Color	Buy
1	low	average	average	< 1000	Blue	Yes
2	high	bad	new	> 1500	Black	No
3	low	average	old	$1000 - 1100$	Red	No
4	average	good	new	$1200 - 1300$	White	Yes
5	high	good	new	$1200 - 1300$	Black	Yes

Table 2.1: Features of a second hand car.

2.1.1 Propositional Representation

We talk about propositional representation, when a problem can be represented by a fixed number of attributes, each of which represents a specific feature of the problem. Let us illustrate this case through an example. Suppose we want to buy a second hand car. If we are not experts in cars, we may take into consideration only a limited number of features, for example the price, the general condition, the age, the power of the engine and the color of the car. We may decide whether to buy a car or not basing our decision only on these features. Each car that we see represents an example of our problem, which is to learn the concept of when to buy a car. Each example is described by five attributes, which identify the features we consider. We could go to visit some second hand cars dealers, and collect a number of examples. We can organize the data we have collected like in table 2.1. In this case we have checked five cars. The first car had a low price, average conditions, it was of average age, had a low power and was blue, and we thought the car was a good deal. The other four cars are described in the same way. So each object is described by a limited and fixed number of attributes. We can then infer a rule for buying a car. An example of such a rule could be *if conditions(good or average) and age(new or average) then buy the car.*

2.1.2 First–Order Logic Representation

The motivation for using first–order logic is that for some problems, e.g molecular biology (Muggleton, 1999) and natural language, propositional logic can not represent adequately the data structures.

As an example, consider the molecule represented in figure 2.1. A molecule consists of several atoms, each of which is described by some properties, e.g., the molecular weight of the atom, or the charge of the atom. In addition to

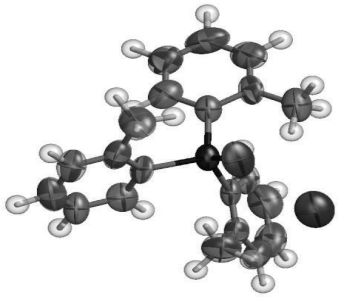


Figure 2.1: A molecule consists of a non-fixed number of connected atoms. Each atom is described by its own set of properties.

properties relative to a single atom, there are relations among atoms. A first kind of relation is represented by links between two atoms. If two atoms are linked to each other we say that there is a bond between them. Atoms can be associated in more than one bond, and there are different types of bonds. Another example of relation among atoms are structures that can exist inside a molecule. A structure can be seen as a relation that involves all the atoms that belong to a particular structure present in the molecule. For example, from the molecule represented in the figure, it is evident that some atoms form a "ring" structure.

If we want to represent such a molecule with propositional logic, we first have to fix a maximum number of attributes, which describe the properties of atoms. Not all the atoms in the molecule possess the same properties. It follows that many of these attributes will not have any value, because they describe properties that are not relative to all the atoms. Then for every relation among atoms there should be an attribute for every possible tuple of the relation. The number of attributes for a relation explodes and is polynomial in the number of available objects. Another problem is that for representing the molecule, one should also fix an order of its atoms. Without an ordering there is an exponential number of equivalent representations of a structure. These problems prohibit an efficient attribute–value representation.

Instead, with first–order logic, we do not have to fix a maximum number of attributes, nor having an attribute for each possible tuple of a relation. Each atom can be described only by the properties relative to it. Relations can be represented by n–ary predicates, whose arguments are the atoms involved in the relation, and possibly the relation type. For example, a bound between two atoms a_1, a_2 could be represented as $bond(a_1, a_2, bound\_type)$.

2.2 ILP

When the language used to express examples, background knowledge and hypotheses is (a fragment of) first–order logic, ICL is called Inductive Logic Programming. ILP can be placed in the intersection between machine learning or data mining and logic programming (Muggleton and Raedt, 1994). ILP shares with the former fields the aim of finding patterns in the data and to develop tools and techniques to induce hypotheses from observations (examples). These patterns can be used to build predictive models or to get some insight of the data. ILP shares with logic programming the use of FOL for the representation of hypotheses and data. Intuitively we can then define the aim of ILP in the following manner:

Definition 2.1 Given are: a set of positive examples E^+, a set of negative examples E^- and background knowledge BK of the concept to be learned, expressed in FOL. Then the aim of ILP is to find a hypothesis H such that H covers all $e \in E^+$ and none of the $e \in E^-$. H is a logic program.

The basic components of FOL are called terms. Terms can be constants, variables or functions. A constant is a name that denotes some particular object in some domain. For example "4" is a constant that denotes the number four in the domain of natural numbers. A variable is a name that can denote any object of a domain. A function symbol denotes a function of arity n taking n arguments from a domain and returning one object of the domain. For example if f is an arbitrary function symbol of arity n and $t_1, \ldots, t_n$ are terms of the same domain, then $f(t_1, \ldots, t_n)$ is a term indicating a function. In addition to terms we have *predicate symbols*. A predicate symbol stands for the name of a relationship between objects. Each predicate symbol has an associated arity.

Definition 2.2 Let P be a predicate symbol of arity n and $t_1, \ldots, t_n$ terms. Then $P(t_1, \ldots, t_n)$ and $\neg P(t_1, \ldots, t_n)$ are literals. $t_1, \ldots, t_n$ are called the arguments of the literal.

Literals can be positives or negatives. For example, $P(a, b)$ is a positive literal, which is true if $P(a, b)$ is true, while $\neg P(a, b)$ is a negative literal, which is true if $P(a, b)$ is false. We refer to a positive literal also as an *atom*.

In this thesis, we consider hypotheses which are logic programs. A logic program is defined in the following way:

Definition 2.3 A logic program is a finite set of Horn clauses.

Definition 2.4 A Horn clause is a clause of the form $A \leftarrow L_1, \ldots, L_n$, where A is an atom and $L_1, \ldots, L_n$ are literals.

In the sequel we consider clauses containing only atoms in the body (so no negation). We say that the part to the left of the arrow is the head of the clause, while the part on the right of the arrow is the body of the clause. Moreover

if the arguments of a clause literals are all ground terms we say that the clause is a *ground* clause. If a clause consists of only the head it is called *fact*. A fact can be ground , e.g., $P(a, b)$., or not ground, e.g., $P(X, c)$.. The first fact states that the object a is in some relation, identified by the predicate symbol P, with another object b, while the second fact states that every object of the domain is in relation with the object c, and we can express this as $\forall X(P(X, c))$. The symbol $\forall$ is called universal quantifier, and the combination $\forall X$ is read *for every object X*. In this thesis, E^+, E^- and BK are sets of ground facts.

A clause has two interpretations, a declarative interpretation (universally quantified FOL implication), which defines the meaning of the clause, and a procedural one, which defines how to solve the clause.

Example 2.1 The declarative interpretation of the clause $p(X, Y) \leftarrow r(X, Z), q(Y, a)$ is:

$$\forall X, Y, Z(r(X, Z), q(Y, a) \rightarrow p(X, Y))$$

and its procedural interpretation is:

in order to solve $p(X, Y)$ *solve* $r(X, Z)$ *and* $q(Y, a)$.

Example 2.2 The following is a logic program:

1. $append([], Y, Y)$.
2. $append([X|X_s], Y, [X|Z_s]) \leftarrow append(X_s, Y, Z_s)$.

This logic program is formed by two clauses. Clause 1 is a fact. $append([X|X_s], Y, [X|Z_s])$ is the head of the clause 2, while $append(X_s, Y, Z_s)$ is its body. $append$ is the only predicate symbol of this logic program. X, Y, X_s, Z_s are variables.

During the learning process we will often need to check whether or not an induced clause covers an example. In the same way, at the end of the learning process we have to check if a whole logic program covers an example. To this aim the programming language Prolog is used. Prolog is an implementation of the logic programming paradigm. In the following we will see how this can be done, but first we have to introduce some more notions of FOL. The first notion we need is the notion of substitution. A substitution is used for instantiating a variable to a particular term of the domain.

Definition 2.5 A *substitution* $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ is a finite mapping from variables to terms that assign to each variable X_i a term t_i, $t_i \neq X_i$, $1 \leq i \leq n$.

Example 2.3 $\theta_1 = \{X/a, Y/b\}$, $\theta_2 = \{Z/tom, X/Y\}$ are examples of substitutions.

Applying a substitution θ to a term t, denoted as $t\theta$, is the result of the simultaneous replacement of each occurrence of a variable in t appearing also in θ with the correspondent term.

Example 2.4 Let $t = f(X,Y)$ and $\theta = \{X/Y, Y/X\}$, then $t\theta = f(Y,X)$. If the replacement were not simultaneous we would obtain the wrong result $f(X,X)$.

Having two literals, it is sometimes possible to render them equal with an application of a substitution. If such a substitution exists, it is called *unifier* . In general there can be many unifiers, and among them a most general one can be identified.

Definition 2.6 Let θ_1, θ_2 be two substitutions. Then we say that θ_1 is more general than θ_2, $\theta_1 \preceq \theta_2$, if there exists a substitution θ_3 such that $\theta_1\theta_3 = \theta_2$.

Definition 2.7 Let L_1 and L_2 be two literals, θ a substitution. We say that θ is a unifier for L_1 and L_2 iff $L_1\theta = L_2\theta$. We also say that L_1 and L_2 are unifiable via θ. θ is the most general unifier (*mgu*) if θ is more general of all the other unifiers of L_1 and L_2.

Example 2.5 Let L_1 and L_2 be $p(X,b)$ and $p(a,b)$ respectively. Then $\theta = \{X/a\}$ is a unifier for L_1 and L_2. Moreover θ is the mgu for L_1 and L_2.

We can question a logic program through the use of queries:

Definition 2.8 A query to a logic program is a clause of the form $\leftarrow L_1 \ldots L_n$ where L_i, $1 \leq i \leq n$, are literals.

A query $\leftarrow P(X,Y)$ can be interpreted as the inquiry $\exists X,Y : P(X,Y)$.? When we pose a query to a logic program, a resolution procedure, called SLD (Selection rule driven Linear resolution for Definite clauses) is applied in order to verify if the query is satisfied by the logic program. The SLD resolution uses the procedural interpretation of the clauses forming the logic program for looking if the query has a successful derivation. If such a derivation exists then the answer will be "yes" otherwise the query fails and the answer will be "no". A SLD derivation is a sequence of SLD derivation steps. If we pose a query $\leftarrow L_1, \ldots, L_n$ to a logic program LP, then a derivation step will consist of the following operations:

1. select an atom L_i, $1 \leq i \leq n$, in the query;
2. select a clause C in LP such that its head can be unified with L_i;
3. select the *mgu* for the query and the head of C;
4. replace L_i in the query with the body of the clause, and apply *mgu* to the resulting query;

In steps 1 and 2, the order in which atoms in the query have to be solved (selection rule) and the order in which clauses of the logic program are used in the derivation need to be specified in order to render the resolution deterministic. If the derivation ends with the empty clause, denoted by $\Box$, then it

is a successful derivation. If the derivation ends with a query in which the selected atom is not unifiable with any clause in the logic program, then it is a derivation of failure. A derivation can be also infinite.

In general, if LP is a logic program from which a query C can be derived in zero or more resolution steps, then we denote this by $LP \vdash C$.

An important property of resolution is that only logical consequences can be derived. This results is known as *soundness* of resolution. In general, a formula F logically implies another formula G whenever any model for F is also a model for G, which we denote by $F \models G$. A model for a formula is an interpretation of the logical language under consideration that makes the formula true. For Horn clauses, we can restrict our attention to so–called least Herbrand models. Every Herbrand model M for a logic program LP determines a set of ground facts that are true in M. The least Herbrand model of a logic program LP, denoted as M_{LP} is the unique set that contains exactly all ground atoms that are true in all Herbrand models for LP. Thus LP logically implies a ground atom A ($LP \models A$) iff M_{LP} contains A.

Once we have defined a selection rule, the totality of SLD derivations for a given query and logic program can be represented by a SLD tree. Each branch of a SLD tree is a SLD derivation via the selection rule. The nodes of the tree are queries with a selected literal. Each node in the tree has exactly one son for each clause that unifies with the selected literal of the query contained in the node.

In Prolog the procedural aspect is implemented using a depth–first search strategy through the clauses defined by a logic program, and by choosing always the first unresolved literal in the query. Prolog builds all possible derivations for the query until it finds a successful one, or until all possible derivations have been tried. In the latter case the query fails.

Example 2.6 Suppose that LP is the following simple logic program:

1. $father(X, Y) \leftarrow parent(X, Y), male(X).$
2. $parent(tom, bill).$
3. $parent(jack, eve).$
4. $parent(eve, bill).$
5. $male(tom).$
6. $male(bill).$
7. $male(jack).$
8. $female(eve).$

and we pose the query $\leftarrow father(X, bill).$ to LP. Then the first and only successful derivation found by Prolog is:

$$\underline{father(X, bill)} \xrightarrow{\theta_1, c1} \underline{parent(X, bill)}, male(X) \xrightarrow{\theta_2, c2} \underline{male(tom)} \xrightarrow{\theta_3, c3} \square$$

where the selected literal in each step of the derivation is underlined, the selected clause is shown like c_i, $1 \leq i \leq 8$, and $\theta_1 = \{Y/bill\}$, $\theta_2 = \{X/tom\}$,

$\theta_3 = \{\}$. $\theta_1\theta_2\theta_3 = \{X/tom, Y/bill\}$ is the computed answer substitution. In this case $father(X, bill)\theta_1\theta_2\theta_3 = father(tom, bill)$ is the computed instance of the query. Note that if we exchange the second and the fourth clause then the first derivation found by Prolog for the query will be of failure.

In the same way, we can verify that also the ground query $\leftarrow father(tom, bill).$ has a successful derivation ($LP \cup \{\leftarrow father(tom, bill).\} \vdash \square$) , while the query $\leftarrow father(eve, bill).$ fails. Finally, if we want to know who is the father of who we pose the query $\leftarrow father(X, Y).$. The computed answer substitutions will be $\{X/tom, Y/bill\}$ and $\{X/jack, Y/eve\}$.

We now have all the instruments for verifying the conditions given in definition 2.1. We can invoke Prolog every time we need to check if a logic program covers an example by posing queries. If Prolog finds a successful derivation for the query than the example is covered, otherwise it is not. Here we exploit the *completeness* of the resolution, that says that if LP is a logic program and A a ground fact, then $LP \models A \; iff \; LP \cup \{\leftarrow A\} \vdash \square$.

Example 2.7 If C is the first clause of the logic program shown in example 2.6, and BK is formed by the other facts of the same logic program, then we have seen that the example $father(tom, bill)$ is covered by C, because Prolog found a successful derivation for the query $\leftarrow father(tom, bill).$ posed to the logic program formed by the union of C and BK. In the same way, we know that C does not cover $e_2 = father(tom, eve)$.

The two main advantages of ILP are:

1. the use of a FOL representation;
2. easy incorporation of a background knowledge of the domain;

The first point is important because, as we have seen, many domains can only be expressed in first–order logic and not in propositional logic.

The second point is important because the use of domain knowledge is essential for achieving intelligent behavior. FOL offers an elegant formalism to represent knowledge and hence to incorporate it in the induction task. The background knowledge is a knowledge common to several examples.

2.2.1 Ordering the Hypothesis Space

The ILP problem can be seen as the problem of searching a hypothesis space for a hypothesis that matches the conditions mentioned in definition 2.1. However, a drawback of a first–order logic representation is that the hypothesis space associated to this representation is usually much larger than the search space associated with a propositional representation. This is because the number of first–order logic candidate solutions is much higher than the number of propositional logic candidate solutions. For this reason the hypothesis space is typically limited by a set of inductive biases, as we will see in sections 2.3.1

and 2.3.2 and in chapter 3. Another aspect that is used is an implicit ordering of the hypothesis space. In fact, the search space can be structured with a *general-to-specific* ordering of hypotheses.

Definition 2.9 A hypothesis H_1 is more general than a hypothesis H_2, and H_2 is more specific than H_1, if all the examples covered by H_2 are also covered by H_1.

Example 2.8 Let H_1 be $father(X, Y) \leftarrow parent(X, Y), male(X).$ and H_2 be $father(X, tom) \leftarrow parent(X, tom), male(X)$. H_1 is more general than H_2, because it will cover more examples than H_2. In particular H_1 will cover all the examples covered by H_2. In fact H_2 covers only examples for which the second argument is equal to tom.

Example 2.9 Let H_1 be $p(X, Y)$, H_2 be $p(a, Y)$ and H_3 be $p(a, b)$, then we have that H_1 is more general than H_2 and H_3 and that H_2 is more general that H_3.

We can imagine the hypothesis space structured in this way. For example, if we have only the predicate symbol p of arity two, the two variables X and Y and the two constant a and b, as in the example 2.9 our hypothesis space, consisting of just one atom, can be viewed in figure 2.2 as a lattice with the general to specific ordering.

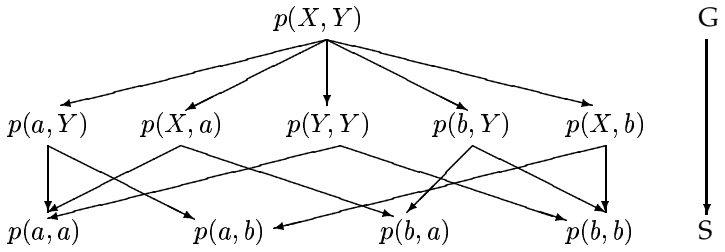


Figure 2.2: Hypothesis space for a simple language. The general to specific order is indicated by the arrow on the right of the tree structure. In the figure an arrow from a literal to another literal means "more general". For instance $p(X, Y) \rightarrow p(a, Y)$ means that $p(X, Y)$ is more general than $p(a, Y)$.

Many systems for ILP exploit this ordering of hypotheses in the operators they use for moving in the search space and for deciding the direction in which the search is performed. The operators vary from system to system, depending on the approach used, the problem to solve, the ideas of the authors and so on. An operator basically receives a hypothesis, changes it in some ways and returns the changed hypothesis. Some systems start the search from a specific hypothesis, which is then generalized during the learning process. This approach is called *bottom-up*. Alternatively a *top-down* approach can be used.

In this case the learning process starts with a general hypothesis which is then specialized to fit the training examples.

An operator used in many ILP systems, is the inverse resolution. To give a flavor of how this operator works, we will describe it for the propositional form, while for details about the inverse resolution in FOL the reader can refer to (Muggleton, 1995). What is done in this method is inverting the resolution rule. Given rules C_1 and C_2, the resolution operator constructs a clause C which is derived from C_1 and C_2. For example, if C_1 is $going\_out \vee staying\_home$ and C_2 is $\neg staying\_home \vee study$ then C will be $going\_out \vee study$. The inverse resolution operator then produces C_2 starting from C_1 and C.

The inverse resolution operator is not deterministic. This means that in general there are multiple choices for C_2. A way for limiting the number of choices is to restrict the representation language to Horn clauses and to use inverse entailment. The idea behind inverse entailment is to change the entailment constraint $BK \wedge H \models e$ into the equivalent form $BK \wedge \neg e \models \neg H$. The previous constraint says that from the background knowledge and the negation of the classification of an example, the negation of a hypothesis explaining the example can be derived. Thus, from the modified constraint one can use a process similar to deduction to derive a hypothesis H. This operator will be used by the system described in section 2.3.2.

Other examples of operators used for moving in the hypothesis space are represented by the operators used by evolutionary systems. We will see examples of evolutionary operators in chapter 3.

2.3 Two Popular ILP Systems

To conclude this chapter, in the next two sections we briefly describe two well known systems for solving ILP problems: FOIL (Quinlan, 1990) and Progol (Muggleton, 1995; Muggleton, 1996). We have chosen to present FOIL because it represents probably the most popular system for ILP, and Progol because of its application to a number of real life ILP problems.

2.3.1 FOIL

FOIL searches the hypothesis space using a top-down search approach and adopts an AQ-like sequential covering algorithm (Michalski et al., 1986). The system first induces a consistent clause and stores it. All the positive examples covered by the learned clause are removed from the training set, and the process is repeated until all positive examples are covered. When a clause needs to be induced, the system employs a hill climbing strategy (for an explanation of hill climbing the reader can refer to e.g., (Russel and Norvig, 1995)). FOIL starts with the most general clause, consisting of a clause with an empty body and head equals to the target predicate. All the arguments of the head are distinct variables. In this way this initial clause classifies all examples as positive. The

clause is then specialized by adding literals to its body. Several literals are considered for this purpose. The literal yielding the best improvement is added to the body. If the clause covers some negative examples then another literal is added. This process is called hill climbing because it proceeds with small steps toward a local best hypothesis. In figure 2.3 a scheme of the algorithm adopted

ALGORITHM($FOIL$)
1 Initialize the clause
2 **while** the clause covers negative examples
3 **do** Find a "good" literal to be added to the clause body;
4 Remove all examples covered by the clause;
5 Add the clause to the emerging concept definition;
6 If there are any uncovered positive examples then go to 1;

Figure 2.3: The scheme of the algorithm adopted by FOIL.

by FOIL is presented. In steps 2 and 3 the hill climbing phase is performed.

The representation language of FOIL is a restricted form of FOL, that omits disjunctive descriptions, and function symbols. Negated literals are allowed in the body of clauses, where the negation is interpreted in a limited way (negation by failure).

The evaluation function used by FOIL to estimate the utility of adding a new literal is based on the number of positive and negative examples covered before and after adding the new literal. More precisely, let C be the clause to which a new literal L has to be added and C' the clause created by adding L to C. The information gain function used is then the following:

$$Info\_gain = sc \cdot (log\frac{p_{C'}}{p_{C'}+n_{C'}} - log\frac{p_C}{p_C+n_C})$$

where $p_C, p_{C'}, n_C, n_{C'}$ is the number of the positive and negative examples covered by C and C', respectively, sc is the number of positive examples covered by C that are still covered after adding L to C.

The add operator considers literals of the following form:

- $P(X_1, X_2, \ldots, X_k)$ and $\neg P(X_1, X_2, \ldots, X_k)$, where X_i's are variables of the clause or new variables;
- $X_i = X_j$ or $X_i \neq X_j$, for variables of the clause;
- $X_i = c$ and $X_i \neq c$, where X_i is a variable in the clause and c is an appropriate constant;
- $X_i \leq X_j$, $X_i > X_j$, $X_i \leq v$ and $X_i > v$, where X_i and X_j are clause variables that can assume numeric values and v is a threshold value chosen by FOIL.

There is a constraint on literals that can be introduced in a clause: at least one variable appearing in the literal to be added must be already present in the clause. Another restriction adopted by FOIL, is motivated by the Occam's razor principle (Blumer et al., 1987). When a clause becomes longer (according to some metric) than the total number of the positive examples that the clause explains, that clause is not considered as a potential part of the hypothesis any more. There is also another bias on the hypothesis space, and it is the upper bound represented by the most general clause initially generated. In fact all the clauses that are generated are more specific than the initial one.

2.3.2 Progol

Progol uses inverse entailment to generate one most specific clause that, together with the background knowledge, entails the observed data. This clause is to bound a top-down search through the hypothesis space with the constraint that only clauses more general than the initial bound are considered.

ALGORITHM($Progol$)
1 If $E = \emptyset$ return BK;
2 Let e be a selected example in E;
3 Construct a most specific clause $\perp$ for e using inverse entailment;
4 Construct a "good" clause C from $\perp$;
5 Add C to BK;
6 Remove from E all the examples that are now covered;
7 Go to 1;

Figure 2.4: Covering algorithm adopted by Progol. The emerging hypotheses are added to the background knowledge and the algorithm is repeated until all the positive examples are covered.

Progol uses a sequential covering algorithm, illustrated in figure 2.4, to carry out its learning task. For each positive example e that is not yet covered, it first searches for a most specific clause, here denoted by $\perp$, which covers e (line 3). For doing this it applies i times the inverse entailment, where i is a parameter specified by the user. In line 4 a A^* strategy is adopted for finding a good clause starting from the most general clause. According to this strategy, a number of clauses are constructed starting from the initial clause. The clause that is considered to be the best is then chosen and the process is repeated.

Progol uses *θ-subsumption* for ordering the hypothesis space. A clause C_1 $\theta-$subsumes a clause C_2 iff there exists a substitution θ such that the set of literals of $C_1\theta$ is contained in the set of literals of C_2 ($C_1\theta \subseteq C_2$), (C_1 is more general than C_2, written also $C_1 \preceq C_2$). The refinement operator maintains the relationship $\square \preceq C \preceq \perp$ for every considered clause C. Thus the search is limited to the bounded sub-lattice $\square \preceq C \preceq \perp$. Since $C \preceq \perp$, there exists a substitution θ such that $C\theta \subseteq \perp$. So for each L in C, there exists a literal

L' in $\perp$ such that $L\theta = L'$. The refinement operator has to keep track of θ and a list of those literals L' in $\perp$ that have a corresponding literal L in C. Any clause C that subsumes $\perp$ corresponds to a subset of literals in $\perp$ with substitutions applied. Among all the refinements the one that is considered the best is chosen, according to an evaluation function, and the process is repeated.

The evaluation function used to measure the goodness of a candidate clause C is:

$$f(C) = p_C - (n_C + lgh_C + h_C)$$

where lgh_C is the length of C, defined as the number of literals in C minus 1, and h_C is the expected number of further atoms that have to be added to the body of the clause. h_C is calculated by inspecting the output variables in the clause and determining whether they have been defined. The output variables are given by a user supplied model.

A first bias on the hypothesis space is represented by the upper bound $\square$ and by the lower bound $\perp$. A second constraint is the use of the head and body mode declarations together with other settings to build the most specific clause. With a mode declaration, the user specifies for each atom used the modality in which an argument can be used. So for example it can be specified that a particular argument is an input variable, or an output variable, or again a particular constant. Progol imposes a restriction upon the placement of input variables. Every input variable in any atom has to be either an input variable in the head of the clause or an output variable in some atom that appeared before in the clause. This imposes a quasi-order on the body atoms and ensures that the clause is logically consistent in its use of input and output variables.

2.4 Conclusions

This chapter provided an brief introduction to ILP. We have first seen how for some classes of problems a propositional representation is not adequate. This motivates the use of first–order logic for representing data. In chapter 9 another example of problem for which a first–order representation is needed is given.

ILP can be seen as a search problem through a hypothesis space, where structures are represented in first–order logic. The objective of the search is to find a hypothesis that covers all the positive examples and none of the negative ones. We have seen how Prolog can be used for checking whether a given hypothesis covers an example or not.

A first–order representation has a great expression power, but this implies that the hypothesis space to search is huge. A strategy for overcoming this is to consider the general-to-specific ordering of the hypothesis space. In this way the hypothesis space can be structured using the concept of generality given in definition 2.9. This ordering allows to search through the hypothesis space in a more efficient way, by means of specialization and generalization operators. The description of two standard ILP algorithms that take advantage of the general-to-specific ordering of the hypothesis space concluded this chapter.

Algorithm	Quality function	Language	Operators
FOIL	Information Gain	FOL without function symbols and disjunctive description	Add literals with at least one variable already in clause
Progol	p_C, n_C, lgh_C, h_C	$\square \preceq C \preceq \perp$	Inverse entailment Refinement operator

Table 2.2: Summary of features of FOIL and Progol. In table p_C and n_C are the number of positive and negative examples covered by C, respectively. lgh_C is the length of C and h_C is an estimate of how many literals still need to be added to C.

In table 2.2, we summarize the main features of the two algorithms. In particular we summarize the features that are considered when assessing the quality of a candidate clause, the language adopted by the two systems and the operators used.

For assessing the quality of a candidate clause, FOIL uses the information gain obtained when a new literal is added to the body of the clause. The literal yielding the best gain is added to the body of the clause. Progol uses a similar strategy. Once the refinement operator has generated a number of candidate solutions the one with higher quality function is chosen and further refined. The quality function used by Progol uses information regarding the coverage of the candidate solution, its length and an estimate of how many refinement steps have to be performed in order to obtain a final clause.

The language adopted by FOIL is a restricted form of first–order logic, where function symbols and disjunctive descriptions are not allowed. The language adopted by Progol vary from clause to clause, and is determined by the most specific clause built with the inverse entailment operator.

Both systems adopt a greedy search strategy for finding good candidate solutions. This gives the systems a good exploitation power, i.e., they are very good at fine-tuning candidate solution, but have rather poor exploration power. This may prevent the systems to escape from local optima.

Chapter 3

Evolutionary Computation

Evolutionary Computation (EC) is a population–based stochastic iterative optimization technique based on the Darwinian concepts of evolution described in the "The origin of species" (Darwin, 1859). Inspired by these principles, like survival of the fittest and selective pressure, EC tackles difficult problems by evolving approximate solutions of an optimization problem inside a computer. An algorithm based on EC is called an evolutionary algorithm (EA). EC has been applied to find solutions of problems in a variety of domains, e.g., planning (Goldberg and Robert, 1985; Fogel, 1988; Jakob et al., 1992), design (Bentley and Corne, 2001; Bentley, 1999; Divina et al., 2003a), scheduling (Davis, 1985; Yamada and Nakano, 1992; Corne et al., 1994), simulation and identification (Roosen and Meyer, 1992; Gehlhaar et al., 1995; Tanaka et al., 1993), control (KrishnaKumar and Goldberg, 1990; Spencer, 1993) and classification (Holland, 1987; Fogel, 1993; Keijzer, 2002).

In this chapter we give some basic notions and principles of EC. The chapter is structured as follows. In section 3.1 we illustrate EC by means of a simple example. In section 3.2 the four paradigms in which EC is usually divided are explained. In section 3.3 we discuss the various components of EC applied to the ICL problem. We start by discussing the representation language and encoding that can be used. We then address the problem of how to evaluate the goodness of an individual, and which aspects of an individual are usually taken into account when assessing the goodness of an individual. Variations operators are then discussed, and some examples are given. In section 3.4 we see how and why the portion of the hypotheses space searched can be limited by means of inductive biases. Section 3.5 addresses the notions of species and niches formation, and the problem of maintaining diversity in the population evolved by an EC system. In section 3.6 we briefly motivate and present the concept of hybrid EC. Section 3.7 presents a summary of the discussed aspects. For a more detailed introduction to EC the reader can refer to (Bäck et al., 2000a; Yao, 2002; Eiben and Smith, 2003a).

3.1 Introduction to Evolutionary Computation

Given an optimization problem, all EAs typically start from a set, called population, of random (candidate) solutions. These solutions are evolved by the repeated selection and variations of more fit solutions, following the principle of the survival of the fittest. We refer to the elements of the population as individuals or as chromosomes. So each individual encodes a candidate solution. Solutions can be encoded in many different ways. A typical example is represented by binary string encoding, where each bit of the string has a particular meaning.

Figure 3.1: Two candidate solutions for the problem of example 3.1.

Example 3.1 Suppose we have a graph made of four nodes, and that each node can be connected to each other. We consider the problem of connecting the nodes in an optimal way, according to some criterion. Two candidate solutions are given in figure 3.1. We could encode these solutions in binary strings in the following way: we fix an order for the possible connections, and associate a bit in the binary string to each possible connection. If a bit relative to a connection is set to 1 then the connection is present in the graph. In total we need 6 bits for representing candidate solutions. We may then consider the following order for connections: (1-2),(1-3),(1-4),(2-3),(2-4),(3-4). The solution depicted on the left hand side of figure 3.1 is then encoded by the string 110011, while 001111 is the binary string relative to the solution proposed on the right hand side of figure 3.1.

The binary strings of example 3.1 represent the genotype of the individuals with phenotype represented by the two graphs of figure 3.1. In general, with the term phenotype we refer to an object forming a possible solution within the original context, while its encoding is called genotype. To each genotype must correspond at most one phenotype, so that the chosen encoding can be inverted, so that genotypes can be decoded.

Individuals are typically selected according to the quality of the solution they represent. To measure the quality of a solution, a fitness function is assigned to each individual of the population. Hence, the better the fitness of an individual, the more possibilities the individual has of being selected for

reproduction and the more parts of its genetic material will be passed on to the next generations of individuals.

Example 3.2 The fitness for individuals of example 3.1 could be a measure of how well the connection criterion is met by individuals.

The selected individuals are modified by means of some variation operators, described in section 3.3.4. From the reproduction phase, new offspring are generated. Offspring compete with the old individuals for a place in the next generation. Typically offspring replace some of the worst individuals in the population, based on the fitness. Another replacement strategy is to use the concept of age, so older individuals are replaced by new individuals.

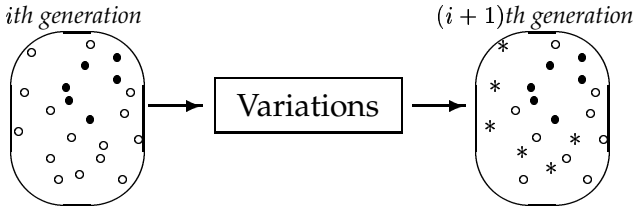


Figure 3.2: In the ith generation selected individuals are represented by black circles. Offspring are inserted in the next generation replacing bad individuals. Offspring are represented by $*$.

A graphical representation of an evolutionary step is given in figure 3.2. The oval on the left hand side represents the old population at the ith generation, while the right hand side oval represents the new population. Individuals in the ith generation are represented by circles, where black circles represent individuals that have been selected for reproduction. These individuals mate by means of some genetic variations and produce offspring, represented in the figure by $*$. In the $(i+1)$th generation the created offspring have replaced some of the old individuals. The process is iterated until a stopping criterion is met. Examples of stopping criteria are setting a maximum number of generations or iterating the process until a good enough individual is generated.

For generating new individuals typically two kind of operators are used: crossover and mutation. In simple terms crossover swaps some genetic material between two or more individuals, while mutation changes a small part of the genetic material of an individual to a new random value.

Example 3.3 Suppose two individuals from the problem presented in example 3.1 are selected, and let these individuals be those represented in figure 3.1

$$\varphi_1 = 110|011$$
$$\varphi_2 = 001|111$$

Then an application of crossover may generate the two new individuals:

Figure 3.3: The two offspring obtained by an application of one-point crossover to the individuals of example 3.1.

$$\varphi_1' = 110|111$$
$$\varphi_2' = 001|011$$

φ_1' encodes the situation depicted on the left hand side of figure 3.3, and φ_2' encodes the situation shown on the right hand side of figure 3.3.

In the above example a so called one-point crossover has been used for creating two new individuals, from two selected individuals, called parents. The operator selects a point inside the two strings, denoted by $|$ in the example, and produces the offspring by exchanging the substrings of the parents. We will see other examples of crossover in section 3.3.4.

The combined application of selection and variation generally leads to improving fitness values throughout generations (Eiben and Smith, 2003b). Evolution is often seen as the process of adaptation to an environment. So fitness can be seen as how the environmental requirements are matched. The better the fitness of an individual the better the individual matches these requirements, and this increases viability, which means that the individual will have more chances to reproduce. So at each generation the population will become more and more adapted to the environment. If we are solving a problem with EC, this means that the population will get closer and closer to the solution.

3.2 Four Paradigms of EC

Four main paradigms of EC can be identified (Eiben and Smith, 2003a):

Evolution Strategies (ES) was introduced in (Rechenberg, 1973). ES typically use an individual representation consisting of a vector of real numbers. ES originally relied most on mutation as main exploratory search operator, but nowadays ES use also crossover.

Evolutionary Programming (EP) was first introduced in (Fogel et al., 1966). EP was originally introduced for developing finite state automata for solving specific problems. Nowadays EP is often used to evolve individuals consisting of real-valued vectors. EP does not use crossover.

Genetic Algorithms (GAs) were introduced by John Holland in (Holland, 1975). GAs typically rely on crossover for exploring the search space. Mutation is considered as a minor operator, and is applied with very low probability. The classic representation used in GAs is a binary string one, however nowadays other kind of representations, such as real-valued strings, are also adopted.

Genetic Programming (GP) was introduced in (Koza, 1992). GP is often described as a variant of GAs. In GP individuals represent some sort of computer programs, consisting not only of data structures, but also of functions applied to those data structures. Individuals typically are tree structures.

Within each paradigm several different algorithms exist, with different features. For this reason the distinction between paradigms is not always so straightforward. More and more methods developed for a particular paradigm are also adopted by other ones.

ALGORITHM($GA - GP$)
1 initialize population;
2 evaluate each individual in population;
3 **repeat**
4 select parents;
5 recombine pairs of parents
6 mutate the resulting offspring;
7 evaluate offspring;
8 insert offspring in the population;
9 **until** (stopping criteria)
10 Extract solution from population;

Figure 3.4: A general scheme of a GA or a GP.

A general scheme of a GA or GP is shown in figure 3.4. In the scheme, the first operation done is the initialization of the population. This can be done at random or with some different strategies. Then each individual of the population needs to be evaluated. Individuals are then evolved (the **repeat** statement). In step 4 a number of individuals are selected from the population. Selected individuals are allowed to generate offspring. Offspring are generated with the application of crossover and mutation in steps 5 and 6. Both crossover and mutation are applied with a given probability, called crossover and mutation rate respectively. They are then evaluated and inserted in the population. The process is iterated over a number of generations, until a stopping criterion is met.

3.3 Various Components of EC

In the following we address various aspects of EC when used for ICL. In particular we discuss the representation of individuals, how to assess the quality of individuals and the operators that can be used for selecting individuals and moving in the search space.

3.3.1 Representation Language and Encoding

In chapter 2, we have seen how important the choice of a representation language is. Once we have a representation language, we need to decide how to encode candidate solutions. An individual can encode a single rule or a set of rules, e.g., a logic program. Whatever the representation used, rules need then to be encoded into individuals. At this aim, various solutions can be adopted, e.g binary strings, real-valued strings, tree structures, high level encoding, etc. In chapters 4 and 5 we will see different solutions adopted for encoding rules.

3.3.2 Evaluation of Individuals

In simple terms what characterizes a hypothesis (candidate solution) as good is how well it performs on the training examples and a prediction of how well its behavior will be on unseen examples. For instance, a hypothesis covering several positive examples and no negative examples could be considered as a good hypothesis. A *fitness function* is used to measure the goodness of a hypothesis. Several properties can be used for defining a fitness function, like: *completeness*, *consistency* and *simplicity*.

Definition 3.1 Let H be an hypothesis. H is said to be *complete* iff H covers all the positive examples.

Definition 3.2 Let H be an hypothesis. H is said to be *consistent* iff H does not cover any negative examples.

Completeness and consistency are two properties that almost all evolutionary inductive learning systems incorporate in the fitness function.

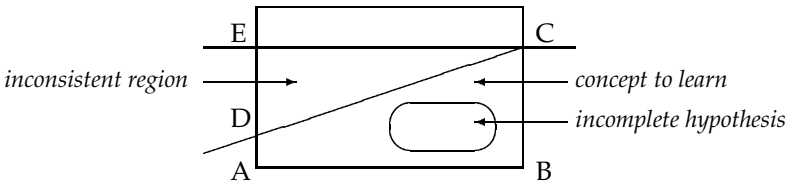


Figure 3.5: Illustration of incompleteness and inconsistency.

Example 3.4 An illustration of both an incomplete and an inconsistent hypothesis is given in figure 3.5. The concept to be found is represented by the area identified by the points $ABCD$. The oval in the figure represents an incomplete but consistent hypothesis, since it fails to cover all the region identified by the target concept, but it does not cover any portion of the region that does not belong to the target concept. Instead, the rectangle $ABCE$ represents a complete but inconsistent hypothesis, since it covers all the region relative to the target concept, but it also covers some portion of the region that does not belong to the target concept.

Simplicity is a concept often used following the Occam's razor principle, which advocates to prefer the simplest hypothesis that fits the data. One rationale explanation for this is that there are fewer short hypotheses than long ones, and so it is less likely that one will find a short hypothesis that coincidentally fits the data. There are many ways for defining simplicity, e.g.,:

Short rules. Prefer shorter rules over longer ones. The length of a rule depends on the representation used, and so the same rule could be considered short by a learner and long by another one.

MDL. This is a more general concept, since it uses a notion of length that does not depend on the particular representation used. According to the *Minimal Description Length* (MDL) principle (Rissanen, 1989) the best model for describing some data, is the one that minimizes the sum of the length of the model and the length of the data given to the model. Here by length we mean the number of bits needed for encoding a model or the data.

Information gain. Information gain (Quinlan, 1986) is a measure of how a change in a hypothesis affects its classification of the examples. This principle when incorporated in the search strategy of a method like in decision trees, biases the search toward shorter rules.

3.3.3 Selection

At each generation, a number of individuals are selected in order to reproduce and generate in this way a new generation. Individuals can be selected in many different ways. Selection is a stochastic process: fitter individuals have higher chances of being selected, but also weak individuals have a chance to become a parent. Examples of selection mechanisms are ranking selection, tournament selection and roulette wheel selection. For more details on selection mechanisms the reader can refer to (Bäck et al., 2000a; Bäck et al., 2000b; Goldberg and Deb, 1991; Blickle and Thiele, 1995).

Ranking selection was proposed in (Baker, 1985). Every individual of the population is given a rank between 0 (less fit) and 1 (most fit). The higher the rank of an individual the higher the probability the individual is selected. In tournament selection n individuals are randomly selected from the population

and the fittest is selected. The parameter n determines the tournament size. A common value for n is 2.

ROULETTE_WHEEL_SELECTION($Individuals$)
1 $size$ = number of $Individuals$
2 $sector$ = array of dimension $size$
3 $tot\_fitness = \sum_{i=0}^{size} fitness(\varphi_i)$
4 $sector_1 = \frac{fitness(\varphi_1)}{tot\_fitness}$
5 **for** $(i > 1) \wedge (i \leq size)$
6 **do** $sector_i = sector_{i-1} + \frac{fitness(\varphi_i)}{tot\_fitness}$
7 $i = i + 1$
8 $random$ = random number between 0 and 1
9 return φ_i such that $sector_{i-1} < random \leq sector_i$

Figure 3.6: Algorithm implementing the roulette wheel selection mechanism. Each entry $sector_i$ of the $sector$ array is associated to the relative individual φ_i.

Also in the roulette wheel mechanism n individuals are randomly selected from the population. A roulette wheel is built, where the sector associated to each of the n selected individuals is proportional to the fitness of the individual. Individuals with higher fitnesses have more probability of being selected, having wider sectors associated to them. The roulette wheel selection mechanism can be implemented by the algorithm shown in figure 3.6. First the sum of all the fitnesses of the individuals is computed. Then a vector, $sector$, is constructed, which represents the sectors of the roulette wheel. A random number is generated, the individual whose sector is the one individuated by the random number is selected. In chapter 4 and in chapter 5 we will see some examples of application of the roulette wheel selection.

Example 3.5 Suppose that three individuals φ_1, φ_2 and φ_3 are randomly selected from the population. Let the fitnesses of the three individuals be the ones shown in figure 3.7. The array $sector$ is then equal to

0.6	0.9	1.0

where 0.6, 0.9 and 1.0 are the upper bounds that delimit each sector. Each entry $sector_i$ is associated to φ_i, $1 \leq i \leq 3$. A random number between 0 and 1 is generated and an individual is selected. It can be seen that φ_1 has more chances of being selected, since its sector is wider. The array can be seen as the roulette wheel shown in figure 3.7, where the dimension of the sectors is equal to the dimension of the sectors $sector_i$. The random seed can be seen as the ball used in a roulette wheel. The roulette wheel is spun and the individual associated to the sector where the ball has stopped is selected.

$$fitness(\varphi_1) = 0.6$$
$$fitness(\varphi_2) = 0.3$$
$$fitness(\varphi_3) = 0.1$$

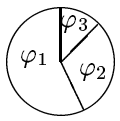


Figure 3.7: An example of roulette wheel for three individuals, φ_1, φ_2 and φ_3.

3.3.4 Variation Operators

From selected individuals, offspring are generated, by means of the application of some variation operators to the parents. The most common variation operators are crossover, or recombination, and mutation.

Crossover operators take two or more individuals as an input, and return two or more offspring. Offspring are obtained by swapping some parts of the parents. The choice of what parts of each parent are combined, and how these parts are combined depend on random drawings. The principle behind crossover is that by mating two individuals with different but desirable features, an offspring that combines both of these features can be created.

Mutation operators receive a single individual. A small part of the individual is changed, and the new mutated individual is returned. Mutation has the role of guaranteeing that the search space is connected.

In the following we will see an example of classical mutation operator. In chapter 5 other mutation operators are introduced. Variation operators can be divided into two classes: standard GA operators and GP operators. The first class of operators, GA operators, act on bit strings. We have already seen an example of crossover, called one-point, in example 3.3. Other examples of crossover operators are the two-point crossover, where two points inside the parents are used, and the uniform crossover, which combines bits sampled uniformly from the two parents.

Example 3.6 An example of two-point crossover is the following: given the two parents

$$\varphi_1 = 111|11111111|111$$
$$\varphi_2 = 000|00000000|000$$

two points are randomly selected inside the strings. The points are denoted by $|$. With the application of the two-point crossover we obtain the two offspring:

$$\varphi_1' = 111|00000000|111$$
$$\varphi_2' = 000|11111111|000$$

Example 3.7 An example of uniform crossover is the following: given the two parents

$$\varphi_1 = 1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|1$$
$$\varphi_2 = 0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0$$

an application of a uniform crossover may produce the following two offspring:

$$\varphi'_1 = 0|1|1|1|0|1|0|1|1|0|0|0|0|1$$
$$\varphi'_2 = 1|0|0|0|1|0|1|0|0|1|1|1|1|0$$

Example 3.8 An example of a classical mutation operator is the following: given the individual

$$\varphi = 1111\underline{1}0000001111$$

a mutation operator can change the fourth bit and the resulting individual is:

$$\varphi' = 1110\underline{0}0000001111$$

The second class of operators, GP operators, act on tree structures. A crossover operator will then create offspring by exchanging subtrees of the parents.

Example 3.9 If the two selected individuals are those shown on the left of the arrow in figure 3.8 then a crossover may produce the two offspring shown on the right of the arrow.

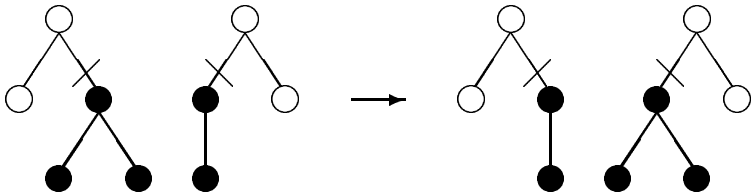


Figure 3.8: Example of GP crossover. The dark part of the two parents are swapped.

Example 3.10 If a selected individual is the one shown on the left of the arrow in figure 3.9, then an application of a GP style mutation may produce the individual shown on the right of the arrow.

3.4 Biases on the Search Space

If we want to assure that a solution is found, it is obvious that the unknown target concept must be contained in the portion of the hypothesis space that is searched. Using a hypothesis space capable of representing every learnable concept could seem the solution, but this would lead to a very large search space. To illustrate this, consider a learner that uses examples described by a set of attributes. In general, in this setting, an unbiased hypothesis space

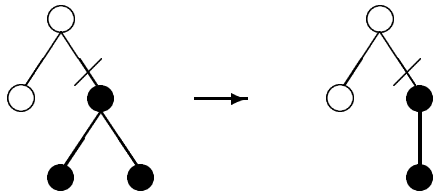


Figure 3.9: Example of GP mutation. The dark part of the left tree has been mutated by deleting one leaf.

contains $2^{|E|}$ possible concepts, where $\mid E \mid$ is the cardinality of the example set. For instance, if a set of attributes can describe 90 different examples of the concept to be learned, then there are 2^{90} distinct target concepts that a learner might be called upon to learn. This is a huge space to search, and for this reason some biases have to be used in order to limit the search to a portion of the hypothesis space.

To limit the size of the search space two main kind of biases are used (Freitas, 2002):

Search bias. This bias determines how an algorithm prefers one hypothesis over others.

Example 3.11 The fitness function is a search bias, because it biases the algorithm towards fitter hypotheses.

Language bias. This kind of bias imposes a constraint on what kind of hypotheses can be represented by the algorithm. The hypothesis space is limited to the resulting set of representable hypotheses.

Example 3.12 If the language used for representing candidate hypothesis is a restricted form of FOL, where only constants and variables are allowed to appear in a clause, then clauses of the form:

$$p(X, Y) \leftarrow q(a, f(Y)), t(f(X), Z).$$

where f is a function symbol, are not part of the language.

3.5 Diversity, Species and Niches

A typical feature of a successful natural system is the presence of different species that can survive in different niches of the environment. The more species can survive in a system, the more successful the system is. Each species can exploit different resources of the system and cooperate or compete with other species. In the same way, within a population maintained by an EA, different species can survive in different niches. A species can be determined in

different ways, as we will see in the following. However the idea of a species in EC is similar to the idea of species in nature: a species differs from another one for some particular features specific of that species.

A motivation for maintaining different species in the population is that in this way computational resources are exploited more effectively by avoiding useless replications and redundancies. Moreover maintaining diversity in the population allows to have individuals spread across the hypothesis space, so that all the areas of the hypothesis space can be searched, and there are no overcrowded regions. This can be seen also as having individuals spread across several niches in the fitness landscape. Problems in which diversity in the population needs to be assured are common in ICL. This because it is often important to assure that the population does not converge to a single super–individual, which covers many examples. Moreover it is important to assure that there are no overcrowded areas in the hypothesis space, but that all the promising regions are explored.

Different methods for achieving species and niches formations, as well as for maintaining diversity in the population, have been proposed. Among these crowding (De Jong, 1975) and sharing function (Goldberg and Richardson, 1987) are two popular methods.

Crowding is a variant of a simple GA with respect to the replacement of older individuals. Instead of substituting the worst individuals in the population, new generated individuals will replace the individuals that are most similar to them, according to a similarity measure. In this way subpopulations are likely to grow, because similar individuals compete with each other.

With sharing function, the fitness of an individual is reduced depending on the number of existing individuals similar to it. In this way the selection probability is modified, with the goal of inhibiting the excessive growth of the genetic pressure of a subpopulation.

The function used for determining the similarity between individuals can act on a genotypic level or on a phenotypic level. At a genotypic level only the structure of the encoding is considered. An example of a function that can be used for determining the similarity between two individuals encoded by bit strings at a genotypic level is the Hamming distance. The Hamming distance is given by the number of 1s resulting by XORing the two bit strings. The Hamming distance can be interpreted as the number of bits which need to be changed to turn one string into the other. At a phenotypic level the distance between two individuals is given by their distance in their semantic domain. Sharing function is a computationally expensive method, since it requires to compute the distance between each pair of individuals of the population. We will see other examples of mechanisms for promoting diversity, as well as species and niches formation in EC applied to ICL in chapter 4.

3.6 Hybrid EC: Memetic Algorithms

Memetic algorithms (MAs) (Moscato, 1989) combine evolutionary-based methods with other techniques that are known to be good for solving a particular class of problems. These techniques usually consist of one or more phases of local search, or of the use of problem-specific information in the operators used in the evolutionary method, as we will see in the following.

The choice for the introduction of MAs is motivated by the fact that EAs are good at exploring the hypothesis space, but are less good at exploitation. This means that EAs can rapidly find good solutions, but they are not as good at "fine-tuning" these solutions to (near) optimality. This is partially due to the stochastic nature of the variations operators adopted in EAs. Moreover in many problems to which EAs are applied, there is a good amount of knowledge and of user experience available. In such cases, benefits can often be obtained when this information is used inside an EA, in the form of specialist operators and/or good solutions, provided that the search is not biased too much that the generation of new solutions is prevented. In these cases the combination of an evolutionary and heuristic method, performs often better than either of the original methods alone.

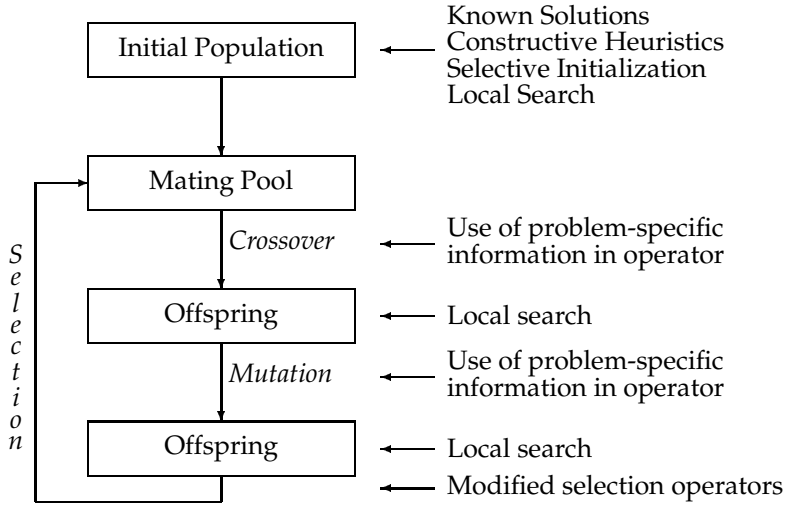


Figure 3.10: Evolutionary phases where knowledge or other operators can be used within the evolutionary cycle.

Figure 3.10 shows in which phases of the evolutionary process knowledge and information about the problem can be incorporated. The first place where knowledge can be incorporated is the initialization of the population. EAs typically initialize the population to random candidate solutions. The initial population could be instead initialized to some known good solutions. These solutions could be known beforehand or be the results of some heuristics. Local

search can be also applied to randomly generated solutions, so that the initial population consists of a set of points that are locally optimal with respect to the operator used in the local search. Local search can be described as an iterative process of examining the set of points in the neighborhood of the current solution, and replacing it with a better neighbor, if one exist. Another way , called selective initialization, is to generate a large set of random solutions and then select a subset of these solutions.

Initializing the population to non-random solutions can have interesting benefits. Using existing solutions can prevent wasting of computational efforts needed to reach already known solutions. Another advantage is that an initial population of non-random solutions can bias the search towards regions of the search space containing good solutions, increasing in this way the effectiveness of the search.

"Intelligent" crossover and mutation operator can be used instead of classical evolutionary variation operators. The "intelligent" operators can exploit problem- or instance-specific information in order to produce better offspring. An example of "intelligent" operator is the mutation operator proposed in (Divina et al., 2003a). A GA is used for evolving a set of Postscript instructions (a set of folds), that when sent to a printer, printed on a sheet of paper (and folded according to the instructions), result in a "flying form" capable of staying in the air for the maximum amount of time possible. A mutation operator used in that GA exchanges the order of the folds, exploiting the information on how single folds are encoded into the genotype. The operator exploits in this way some knowledge about the problem. Other examples of "intelligent" operators are given in chapter 5.

The most common use of hybridization within EAs, is the application of one or more phases of local search acting on the offspring created by mutation or crossover. In this way there is the chance that if variation operators generate a solution that is close to the optimum, this solution can be improved in such a way that the optimum can be reached.

Information about the problem can be used also in the selection operator. This can promote diversity in the population, avoiding in this way the premature convergence around some suboptimal point of the search space. Premature convergence of the population is a problem that afflicts many MAs. This is due to the application of local search. If the local search phase continues until each point has been moved to a local optimum, then this leads to an inevitable loss of diversity within the population. An example of selection operator that incorporates knowledge about the problem is given in chapter 5.

A last point, not listed in the figure, where knowledge about the problem can be easily incorporated, is in the decoding of individuals. Here knowledge can be applied in order to perform an intelligent decoding, producing better phenotypes.

3.7 Conclusions

The aim of this chapter was to provide the reader with the basic notions of evolutionary computation, and in particular to provide the notions on various aspects of EC when used for ICL.

EAs work by the repeated application of selection, recombination and mutation of fit individuals. What distinguishes EAs from local search algorithms, is the fact that EAs do not operate only on one candidate solution. Instead EAs operate on a number of candidate solutions, called population, at the same time. The use of a population determines a nonuniform probability distribution function. This function governs the generation of new candidate solutions starting from a given population. The probability distribution function reflects possible interactions among candidate solutions.

Maintaining a population not only gives the possibility to EAs of escaping from local optima, it also gives the possibility to cope with large and discontinuous search spaces. This is important for ILP, where the hypothesis space to search is huge and where noise in the data can be present.

Chapter 4

EC applied to ILP

Evolutionary computation has proved to be successful in solving comparatively hard optimization problems, as well as problems like ICL (Goldberg, 1989; De Jong et al., 1993). Moreover GAs, GP and ES have some features that render them an attractive alternative to greedy rule induction algorithms for tackling ILP problems. First, GAs, GP and ES have the capability of escaping from local optima, while greedy algorithms may not show this ability. Secondly, these methods have an intrinsic parallelism and can therefore exploit parallel machines much more easily than classical search algorithms. Finally these methods tend to cope better than greedy rule induction algorithms when there is interaction among arguments (Freitas, 2002).

Depending on the representation used, two major approaches can be individuated: the *Pittsburgh* and the *Michigan* approach, so called because they were first introduced by research groups at the Pittsburgh's and Michigan's university, respectively. In the former case each individual encodes a whole solution (in this case a logic program), while in the latter case an individual encodes a part of the solution (e.g., a single clause). Both approaches present advantages and drawbacks. The Pittsburgh approach allows an easier control of the genetic search, but introduces a large redundancy that can lead to hard to manage populations and to individuals of enormous size. The Michigan approach, on the other hand, allows for cooperation and competition between different individuals, hence reduces redundancy, but requires sophisticated strategies, like co-evolution, for coping with the presence in the population of super individuals. Moreover, in the Pittsburgh approach, at the end of the evolutionary process, the best individual represents the solution to the problem, while in the Michigan approach, some mechanisms have to be used for extracting a subset of the population forming the solution to the problem.

A number of systems based on EC have been proposed to solve ILP problems. In the following we give a brief description of some of these systems. The order in which the systems are presented reflects the complexity of the representation adopted, starting from binary strings and ending with a tree representation. For an extended overview of evolutionary systems for ILP the

reader can refer to (Divina, 2001a).

4.1 REGAL

REGAL (RElational Genetic Algorithm Learner) (Neri and Saitta, 1995; Giordana and Neri, 1996) exploits the explicit parallelism of GAs. In fact it consists of a network of genetic nodes fully interconnected and exchanging individuals at each generation. Each genetic node performs a GA. A supervisor node is used in order to coordinate these subpopulations. The system adopts a Michigan approach, each individual encodes a partial solution, i.e., a clause.

The language used by REGAL is an intermediate between VL_2 and VL_{21} (Michalski et al., 1983; Michalski et al., 1986), in which terms can be variables or disjunctions of constants, and negation occurs in a restricted form. An atomic formula of arity n has the form $P(X_1, \ldots, X_n, K)$, where $X_1, \ldots, X_n$ are variables and K is a disjunction of constant terms, denoted by $[v_1, \ldots, v_m]$, or the negation of such a disjunction. For example, these are well formed formulas: $shape(X_1, [triangle, rectangle])$, $distance(X_1, X_2, [1, 2, 3])$, $name(X_1, \neg[Caio, Tizio])$ meaning, e.g., for the first rule, that the shape of X_1 is either a triangle or a rectangle.

Before introducing how individuals are actually encoded by REGAL, we first have to introduce the concept of language template used by REGAL. Informally, the template is a formula Λ belonging to the language, such that every admissible conjunctive concept description can be obtained from Λ by deleting some constants from the internal disjunctions occurring in it. The predicates in the template can be divided in predicates in *completed form* and those not in completed form.

Definition 4.1 A predicate $P(X_1, \ldots, X_n, [v_1, \ldots, v_m])$ is in completed form if the set $[v_1, \ldots, v_m]$, which constitutes its internal disjunction, is such that the predicate can be satisfied for any binding of the variables $X_1, \ldots, X_n$.

For instance, $shape(X_1, [triangle, rectangle, *])$ is in completed form. The symbol * means "everything which does not appear in the internal disjunction". A language template Λ must contain at least one predicate in completed form. Indeed, given a language template, the search space explored by REGAL is restricted to the set $H(\Lambda)$ of formulas that can be obtained by deleting some constants from the completed terms occurring in Λ. This is because predicates not in completed form have the role of constraints and must be satisfied by the specific binding chosen for the variables in Λ. On the other hand, predicates in completed form are used to define the search space. Deleting a constant from a completed term makes the term more specific.

Since the search space is limited to $H(\Lambda)$, only predicates in completed form need to be processed, and so encoded. REGAL uses bit strings for this purpose, where a string is divided into substrings. Each substring corresponds to a literal, in the same order as they appear in the language template. Each bit

$$
\begin{array}{rcl}
\mathbf{\Lambda} & = & weight(X,[3,4,5]) \wedge color(X,[red,blue,*]) \wedge shape(X, \\
 & & [square,triangle,circle,*]) \wedge far(X,Y,[1,2,3,4,5,*]) \\
\mathbf{\Lambda_s} & = & color(X,[red,blue,*]) \wedge shape(X,[square,triangle,circle,*]) \\
 & & \wedge far(X,Y,[1,2,3,4,5,*]) \\
\varphi_\mathbf{1} & = & weight(X,[3,4,5]) \wedge color(X,[red]) \wedge shape(X,\neg[square,circle]) \\
 & & \wedge far(X,Y,[1,2]) \\
\varphi_\mathbf{2} & = & weight(X,[3,4,5]) \wedge color(X,\neg[red]) \wedge far(X,Y,[1,2])
\end{array}
$$

$s(\Lambda_s) \rightarrow$	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\varphi_1 \rightarrow$	1	0	0	0	1	0	1	1	1	0	0	0	0	0
$\varphi_2 \rightarrow$	0	1	1	1	1	1	1	1	1	0	0	0	0	0

Figure 4.1: In the figure Λ_s is the subset of Λ consisting of the predicates in completed form. The bit strings are divided in substrings each of them corresponding to a predicate in completed form, appearing in the same order than in Λ. *weight* is not encoded since it is not in completed form.

corresponds to a term. If the bit corresponding to a given term v_i in a predicate P is set to 1, then it means that v_i belongs to the current internal disjunction, whereas, if it is set to 0 it does not belong to the internal disjunction. An example of a language template and of the representation of formulas is given in figure 4.1. The template is supplied by the user. This implies that the user has some knowledge about the form of the rules that have to be learned. We will see different approaches in section 4.4 and in chapter 5.

When the system evaluates a formula on an example, each variable in the formula has to be bound to some object in the description of the example. Then the predicates occurring in the formula are evaluated on the basis of the attributes of the object bound to their variables. A formula is said to be true on an example iff there exists at least one choice such that all the predicates occurring in the formula are true. The user has to specify how to evaluate the semantics of the predicates before starting to run REGAL on a specific application. The fitness of an individual φ, to be maximized, is given by the function $f(\varphi) = f(z, n_\varphi) = (1 + Az)e^{-n_\varphi}$, where z is a measure of the simplicity [1] of the formula, A is a user tunable parameter with default value of 0.1 and n_φ is the number of negative examples covered by φ.

Individuals are selected for reproduction by means of the *Universal Suffrage* (*US*) selection mechanism (Giordana and Neri, 1996). This selection mechanism represents the base of the selection mechanisms used in the system presented in chapter 5. The US selection mechanism works as follow:

[1] Namely z is the number of 1s in the string divided by the length of the string.

NODAL GENETIC ALGORITHM(Node ν)

```
1  Initialize the population A_ν(0) and evaluate it
2  while not solve
3  do receive μ· | A_ν(t) | individuals from the network and store
4           them in A_net(t)
5     Select B_ν(t) from A_ν(t) ∪ A_net(t) with the US operator
6     Recombine B_ν(t) using crossover and mutation
7     Update A_ν(t) and A_net(t) with the new individuals in B_ν(t)
8     Send A_net(t) on the network
9     Send the status to the supervisor
10    Check for messages from the supervisor
```

Figure 4.2: Genetic algorithm used by a node ν in the distributed version of REGAL. The algorithm is repeated until the node receives a *solve* signal from the supervisor.

1. the operator randomly selects n positive examples e_i, $1 \leq i \leq n$;

2. for each e_i an individual is selected as follows. A roulette wheel is performed among individuals covering e_i. The dimension of the sector associated to each individual is proportional to its fitness. The winner of the tournament is selected for reproduction. If e_i is not covered then a new individual covering e_i is created using a seed operator.

Example 4.1 Suppose that there are five positive examples e_i, $1 \leq i \leq 5$, and that example e_3 is randomly selected in the first step of the US selection operator. Suppose moreover that e_3 is covered by three individuals φ_1, φ_2 and φ_3, with fitnesses $f(\varphi_1) = 0.6$ $f(\varphi_2) = 0.3$ $f(\varphi_3) = 0.1$. Here we suppose that the fitnesses are normalized. Then the roulette wheel built for the three individuals looks like the one given by figure 3.7, and φ_1 has the greater possibilities of being selected.

Species formation is achieved with the US selection operator, in fact only individuals covering the same examples compete with each other.

REGAL adopts four crossover operators: the classical two-point and uniform crossovers described in chapter 3, and a *generalizing* and *specializing* crossovers. The generalizing crossover works by OR-ing some selected substrings of the parents, while the specializing crossover works by AND-ing them. The probability of applying the first two classical crossovers is higher when the two selected individuals have a low fitness. Conversely, the higher the fitness the more likely is to apply the other two crossovers. This choice is justified by the observation that two-point and uniform crossovers have an high exploration power, while the generalizing and specializing crossover can be used for refining individuals that are already good. The mutation operator is a classical bit mutation operator, and can affect all the bits of the string.

A first bias for limiting the hypothesis space is represented by the language template. The set of examples that is assigned to a particular node represents another bias. A node will develop individuals that belong to the species determined by the examples assigned to the node.

In figure 4.2 a scheme of the genetic algorithm for a node ν is presented. In line 3 the node receives a number of individuals from the network, these individuals will be used for avoiding the lethal mating problem, i.e., matings that are bound to produce bad offspring. The execution will end when the node receives a *solve* signal from the supervisor. During the learning process the supervisor periodically receives and stores the best solution found by each genetical node.

At the end of the learning process, since REGAL adopts a Michigan approach, a solution has to be extracted from the rules received by the supervisor. For this purpose first the set E_H^+ is constructed, as the union of all positive examples covered by the received clauses. The clauses are sorted in decreasing order according to $\pi(C) = p_C \times f(C)$. The minimum number of best clauses able to cover E_H^+ represent the solution.

4.2 G-NET

G-NET (Genetic Network) (Anglano et al., 1998) is a descendant of REGAL. As its predecessor, G-NET is also a distributed system, with a collection of genetic nodes and a supervisor.

G-NET adopts a co-evolution strategy by means of two algorithms. The first algorithm computes a global concept description out of the best hypotheses emerged in various genetic nodes. The second algorithm computes the assignment of the positive examples to the genetic nodes. The assignment strategy consists in addressing the search on positive examples that are covered by poor hypotheses, without omitting to continue the refinement of the other hypotheses.

G-NET is based on the same theory of species and niches formation and on the same language adopted by REGAL. G-NET differs from REGAL by the fitness function. In fact G-NET uses two functions. A first one is used at a global level, while a second function is used at local level, so for evaluating a clause in a genetic node. A global hypothesis H is evaluated in the following way:

$$f_G(H) = MDL_{MAX} - MDL(\neg p_H + n_H) - MDL(H)$$

where MDL_{MAX} is the MDL (see section 3.3.2) of the whole learning set and $\neg p_H$ is the number of positive examples not covered by H.

The formula used for evaluating an individual φ at the local level is the following:

$$f_L(\varphi) = MDL_{MAX} - MDL(\varphi) - MDL(\neg p_\varphi) + (f_G(H') - f_G(H))$$

In the above formula H is the current global hypothesis, H' is the hypothesis obtained by adding φ to H.

G-NET adopts three kinds of mutation operators. One of the mutation operators is used in order to generalize an individual, another one is used for the specialization and a third mutation operator is used for creating new clauses, so it can be also seen as a seeding operator. The crossover is a combination of the two-point crossover with a variant of the uniform crossover, modified in order to perform either generalization or specialization of individuals. Both crossover and mutation operators enforce diversity, so that it is assured that in a genetic node there are no equal clauses.

4.3 DOGMA

DOGMA (Domain Oriented Genetic MAchine) (Hekanaho, 1996; Hekanaho, 1998) employs two distinct levels. On the lower level the Michigan approach is adopted, while on a higher level the Pittsburgh approach is used.

On the lowest level the system uses fixed length chromosomes, which are manipulated by crossover and mutation operators. On the higher level chromosomes are combined into genetic families, through some special operators that can merge and break families.

The representation adopted by DOGMA is the same representation used by REGAL.

The fitness function combines two different functions. One is based on the MDL principle, and the other is based on the information gain measure. The total description length of a hypothesis H consists of the hypothesis cost, i.e. the length of the encoding of H, and the exception cost, which is the encoding of the data that is erroneously classified by H. The unit of length used is a binary digit. To turn the MDL principle (see section 3.3.2) into a fitness function, the MDL of the current hypothesis H is compared against the total exception length with the following function:

$$f_{MDL}(H, E) = 1 - \frac{MDL(H,E)}{W_\emptyset \times MDL(H_\emptyset, E)}$$

In the above formula, $MDL(H_\emptyset, E)$ stands for the total exception length, i.e. the description length of an empty hypothesis that covers no examples. $W_\emptyset$ is a weight factor that is used to guarantee that even fairly bad hypotheses have a positive fitness. This function alone can not be used as a fitness function. In fact the function under rates hypotheses that are almost consistent and very incomplete. This would lead to a prevalence of fairly large, but very incomplete clauses, since these are mostly preferred by the function to fairly small, but almost consistent clauses. In this way the population would become overly general and very inconsistent. For this reason, the function based on information gain is used. This function promotes small and almost consistent clauses. The information gain of a hypothesis H compared to another hypothesis H_{def} measures how much information is gained in the distribution of correctly and

incorrectly classified positive examples of H compared to the distribution of H_{def}. The fitness function based on the information gain uses this gain measure:

$$Gain(H_{def}, H, E) = log_b(p_H + 1) \times (Info(H_{def}, E) - Info(H, E)),$$

where $b \geq 1$ (default value 1.2) and $Info(H, E) = -log\frac{p_H}{p_H + n_H}$. Hypotheses are compared against H_{def}, a default hypothesis that classifies all examples as positives. The fitness function based of the information gain is then defined as follows:

$$f_G(H, E) = W_G \times \frac{Gain(H_{def}, H, E)}{Gain(H_{def}, H_{max}, E)}$$

where $W_G > 0$ is a tunable parameter, and H_{max} is a hypothetical hypothesis that correctly classifies all examples. Finally the fitness function used by DOGMA is the following:

$$f_{MG} = min(f_{MDL}(H, E), f_G(H, E))$$

that chooses the minimum between f_{MDL} and f_G.

To enhance diversity and to separate different kinds of clauses, the system uses speciation of chromosomes. This can be done randomly or by dividing individuals into species according to which parts of the background knowledge they may use. Speciation has three applications in the system. First it is used for controlling the mating of chromosomes of different species. Secondly, speciation can control what part of the background knowledge individuals can use. Finally, speciation is used when merging chromosomes into families. Chromosomes of the same species cannot be merged in the same family.

DOGMA uses the crossover operators used by REGAL, a classical mutation operator, and a seeding operator which, given a randomly selected example, randomly creates a bit string and then adjusts it in order to cover that example. The remaining operators work on the family level. A *break* operator splits randomly a family into two separate families. In opposition to the break operator, a *join* operator joins two families into one. If there are two chromosomes of the same species then one of them is deleted. In addition to these operators, a *make-family* operator is used for forming families by selecting useful chromosomes of different species from the population. The order in which the operators are currently applied is given in figure 4.3.

DOGMA follows the metaphor of competing families by keeping genetic operators, such as crossover and mutation, working on the lower level, and by building good blocks of chromosomes, while lifting selection and replacement to the family level. Fitness is also lifted to the higher level.

4.4 SIA01

SIA01 (Supervised Inductive Algorithm version 01) (Augier et al., 1995) uses the sequential covering principle developed in AQ (Michalski et al., 1986). The

MAKE-NEXT-GENERATION(P)
1 $P_S \leftarrow$ Select families in P
2 $P_{S'} \leftarrow$ Mate chromosomes P_S
3 $P_X \leftarrow$ Crossover$P_{S'}$
4 $P_M \leftarrow$ Mutate P_X
5 $P_B \leftarrow$ Break families P_M
6 $P_J \leftarrow$ Join families P_B
7 $P_U \leftarrow P_J \cup$ Make families P
8 $P_E \leftarrow$ Evaluate P_U
9 $P' \leftarrow$ Replace families (P, P_E)
10 **return** P'

Figure 4.3: Algorithm used by DOGMA for the creation of a new population P' starting from an old population P.

system adopts a bottom-up approach. For creating the initial clause, SIA01 randomly chooses an uncovered positive example and uses it as a seed. Then it finds the best generalization of this clause according to some evaluation criterion. This is done by means of a GA. To obtain a new generation the algorithm applies to each individual in the population a genetic operator, and then the newly created individual may be inserted in the population. The size of the population can grow in this way until a certain bound is reached. A scheme of the GA used for searching the best clause is represented in figure 4.4.

A high level representation, similar to the one adopted by the system introduced in this thesis (chapter 5), is used for encoding clauses. SIA01 uses predicate symbols and their arguments as genes.

For instance the clause $Pyramid(X) \leftarrow Color(X, yellow), Support(Y, X).$, will be encoded in the following individual:

Pyramid	X	Color	X	yellow	Support	Y	X

The fitness function used takes into consideration the consistency of the clause, its completeness, its syntactic generality and some user's preferences:

$$f(C) = \begin{cases} (1-\alpha-\beta) \times CM + \alpha S + \beta A & if \quad CN > 1 - N \\ 0 & otherwise \end{cases}$$

In the above formula CM is the absolute completeness and it is defined as $\frac{p_\varphi}{|E^+|}$, where $\mid E^+ \mid$ is the total number of positive examples. CN is the absolute consistency and it is defined as $\frac{|E^-|-n_\varphi}{|E^-|}$ where $\mid E^- \mid$ is the total number of negative examples. N is the maximum noise tolerated, S is the syntactic generality of φ and A is the clause's appropriateness to the user's preferences. N, A, α and β are user tunable parameters.

A mutation operator and two crossover operators are used for creating new individuals. The mutation operator selects a relevant gene and performs one of the following operations:

GA($SIA01$)
1 *Pop* = *Seed*
2 **repeat**
3 **for** $\forall \varphi \in Pop$
4 **do if** φ has not already produced offspring
5 **then** create one offspring by mutation of φ
6 create two offspring by crossover with φ'
7 put the offspring in Pop'
8 **for** $\forall \varphi \in Pop'$
9 **do if** $\varphi \notin Pop$
10 **then if** $size(Pop) < max$
11 or *fitness* φ *is better than the worst fitness in Pop*
12 **then** insert φ in *Pop*
13 **until** *fitness(best* φ*)* has not changed since last n generations

Figure 4.4: The scheme of the GA adopted by SIA01. φ' is an individual in the population that has not yet produced any offspring.

- if the gene encodes a predicate symbol then change it with a more general predicate symbol, according to the background knowledge. If it is not possible to generalize anymore then drop the predicate. For example the predicate symbol $Pyramid$ could be changed into $Pointed-top$ without modifying the arguments of the predicate;
- if the gene encodes a numerical constant then the mutation can create an interval, or if the gene is already an interval the operator can enlarge it;
- if the gene encodes a symbolic constant, then the operator may create an internal disjunction or generalize an existing disjunction;
- if the gene encodes a symbolic constant this constant can be turn into a variable. This change is reported in the whole individual;
- if the gene encodes a variable the operator may replace it with another variable.

The first crossover, which is used by default, is a restrained one-point crossover. The restriction is that the chosen point in the clause has to be before a predicate. If the seed contains only one predicate then the standard one-point crossover is used.

4.5 GLPS

The Genetic Logic Programming System (GLPS) (Wong and Leung, 1995) is a GP system, where an individual represents a program, following the Pittsburgh approach. The reproduction phase involves selecting a program from

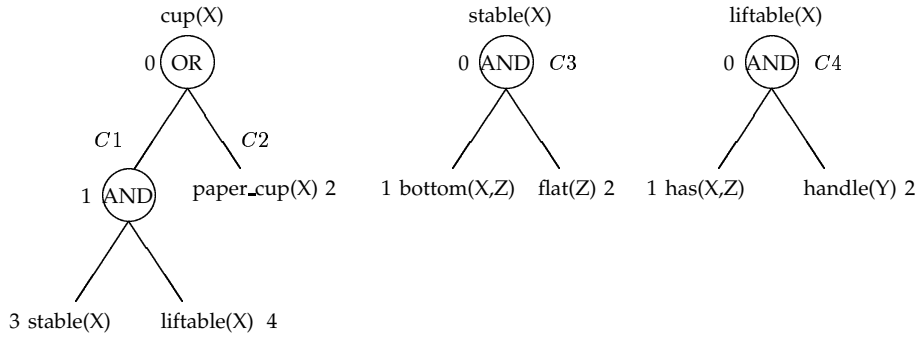


Figure 4.5: A forest of AND-OR trees. The numbers next to each node are the identifier numbers of the nodes.

the current population of programs and allowing it to survive by copying it into the new population. The selection is based either on tournament or on fitness proportional selection, where individuals are sorted according to their fitness and the first n individuals are selected. The system uses crossover to create two offspring from the selected parents. GLPS does not use any mutation operators. After the reproduction and crossover phase, the new generation replaces the old one. Next, GLPS evaluates the population assigning a fitness value to each individual and iterates this process over many generations, until a termination criterion is satisfied.

The system adopts a restriction on the representable clauses: function symbols can not appear in a clause. Logic programs are represented as a forest of AND-OR trees, being the leafs of such trees positive or negative literals containing predicate symbols and terms of the problem domain. For example, figure 4.5 represents the logic program:

$$
\begin{aligned}
C1: &\quad cup(X) \leftarrow stable(X), liftable(X).\\
C2: &\quad cup(X) \leftarrow paper\_cup(X).\\
C3: &\quad stable(X) \leftarrow bottom(X, Z), flat(Z).\\
C4: &\quad liftable(X) \leftarrow has(X, Y), handle(Y).
\end{aligned}
$$

In the figure the identifier of each clause is reported next to the relative tree or branch. For instance, the left most tree represents both clauses $C1$ and $C2$, where the left subtree starting at node 1 is relative to $C1$ and the right subtree is relative to $C2$.

With this representation it is not difficult to generate an initial population randomly. A forest of AND-OR trees can be randomly generated and then the leaves of these trees can be filled with literals of the problem. Another way is to generate the initial population using some other systems, like FOIL.

The fitness function used by GLPS is a weighted sum of the total number of misclassified positive and negative examples. The weight is used for dealing with uneven distribution of positive and negative examples.

An ad-hoc crossover operator is used, which can operate in various modalities: individuals are just copied unchanged to the next generation, individuals

Algorithm	Encoding	Approach	Fitness features
REGAL	bit strings (initial template)	Michigan	con + sim
G-NET	bit strings (initial template)	Michigan	2 functions. con + sim + MDL
DOGMA	bit strings (initial template)	Michigan & Pittsburgh	MDL + Gain
SIA01	high level representation	Michigan	con + com + gen + pref
GLPS	AND-OR trees	Pittsburgh	con + com

Table 4.1: In the table *con* stands for consistency, *com* for completeness, *sim* for simplicity, *pref* for user's preferences, *gen* for syntactic generality. MDL is the Minimum Description Length Principle. Gain is the information gain.

exchange a set of clauses, a number of clauses belonging to a particular rule are exchanged between the individuals, a number of literals belonging to a clause are exchanged.

4.6 Discussion

We end this chapter with a brief comparison of the presented systems for what regards the encoding, the fitness function, the selection operator, the variation operators and biases adopted for limiting the search space. Table 4.1 summarizes the encoding adopted by the systems, the approach used (either Michigan or Pittsburgh) and the properties used in the fitness function. Table 4.2 summarizes the variation operators and the type of selection operator adopted.

Representation REGAL, G-NET and DOGMA use the supplied template to map clauses into bit strings. This implies some knowledge of what the user expects to discover, which cannot be always taken for granted. The use of the initial templates also imposes another limitation. All the rules handled must follow the initial given template, which is constant and cannot change during the learning process. Also with this approach, some problems can arise when dealing with numerical values. In fact the binary representation of the model can become quite long and this may slow down the process.

The bit string representation used by these three systems does not allow them to perform some operations that are typical FOL operations, e.g., changing a constant into a variable. The high level representation adopted by SIA01 is more flexible, where the shape of clauses learned can vary during the learning process. In fact the particular shape of each clause is determined by the positive example used as seed in the initialization phase.

Algorithm	Type of crossover	Mutation	Selection
REGAL	uniform, two-point generalizing, specializing	classic	US
DOGMA	uniform, two-point generalizing, specializing	classic	lifted on family level
G-NET	generalizing, specializing two-point	generalizing specializing	tournament
SIA01	restrained 1-point classic 1-point	4 generalizing modalities	inds that haven't produce offspring
GLPS	reproduction exchange info	none	tournament

Table 4.2: A summary of the characteristics of the various operators adopted by the presented systems. inds means individuals.

GLPS does not require an initial template either, so the shape of the initial clauses is not fixed.

Fitness Function The system that adopts the simplest fitness functions is GLPS. It takes into consideration only the completeness and the consistency of individuals. The function adopted by SIA01 is more elaborated. In addition to completeness and consistency, simplicity is considered, and the user can express some preferences for some type of clauses. However, consistency and completeness are the features that have the biggest influence on the fitness assigned to an individual.

The function used by REGAL considers only simplicity and consistency. Completeness is not considered because the US selection operator already promotes complete individuals. This reduces the complexity of the evaluation. G-NET uses two fitness functions, one used at a local level, in the genetic nodes, and another one used at a global level. G-NET also makes use of the MDL principle in its fitness functions. Unlike REGAL, at a local level G-NET considers also the global behavior of clauses. This is achieved by evaluating how well a clause combines with others in order to form a global solution. This is a good strategy, since G-NET is a distributed system.

DOGMA combines the information gain and the MDL principle. Information gain is used for promoting small and almost consistent clauses. This is done because using only the MDL would result in an under rating of hypotheses that are almost consistent but very incomplete. This would lead DOGMA towards a population with a majority of large and very inconsistent clauses. In this way the population would be too general and very inconsistent.

Selection Operators The US selection operator adopted by REGAL allows the system to achieve species and niches formation without the need of any

distance measure. This because only individuals covering the same examples compete with each other for being selected. Moreover a good coverage of positive examples is encouraged. However, some mechanisms exploiting the distributed implementation of the system, have to be adopted for avoiding the presence of super individuals in the subpopulation evolved in the nodes. In the next chapter we describe a way for avoiding this problem in a non parallel system. G-NET adopts a similar strategy for promoting species, but uses a tournament selection.

In DOGMA the selection is raised to the family level and is simply based on the fitness of the families.

SIA01 adopts a different method of selection, which is not based on the fitness of individuals. In fact at each generation all individuals that have not produced an offspring are selected for reproduction.

GLPS can use either tournament selection or a fitness proportional selection, where n individuals with higher fitness are selected.

Variation Operators REGAL and DOGMA adopt the same variation operators. Two crossovers are used to generalize and specialize individuals. In addition to these, uniform and two-point crossovers are used in order to make bigger steps in the hypothesis space.

G-NET introduces some novel ILP operators. Both the crossover and the mutation operators, can be used in three modalities. For the crossover these modalities are: generalization, specialization and exchanging modality. The latter one is implemented by a classical two-point crossover. The generalization modality tends to be used when both parents are consistent, and otherwise the specialization modality is more likely to be applied. For the mutation a similar strategy is applied. With this strategy when in a node a clause is often chosen the search turns into a stochastic hill climbing.

What is done by these three systems when they generalize or specialize a rule, is basically dropping or adding conditions. This is because the systems adopt an internal disjunction in order to define the values that a variable can assume.

SIA01 has the possibility to perform some operations which are more FOL oriented. The mutation operator used by SIA01 can perform a variety of operations, e.g., changing a numeric constant into an interval or turning a constant into a variable. An interesting feature is that the operators are designed in a way that guarantees that individuals of the population are syntactically different from each other. SIA01 relies mostly on mutation. This is because the high level representation adopted makes the design of an effective crossover operator difficult.

In opposition, GLPS does not make use of any mutation operators, so the reproduction phase is carried out only by the crossover operator, which can exchange rules, clauses or just literals.

Biases REGAL, DOGMA and G-NET limit the hypothesis space by means of an initial template. Only clauses that can be derived from the initial template are considered during the search of a satisfying concept. SIA01 considers clauses limited to the possible generalizations of the initial clause that was built starting from a seed example. Therefore, SIA01 uses different biases for each individual, depending on the example that is used for creating the individual. In this way individuals are not constrained into a fixed shape. GLPS restricts the search to individuals than can be represented with trees having a maximum depth specified by a user tunable parameter.

All the systems presented in this chapter are very good at exploring the search space, but are not as good at exploitation. The exploration power they have gives the systems the capability of efficiently searching the hypothesis space, and the ability of escaping from local optima. However, they are not very good at fine tuning candidate solutions. This is different from what happens in the systems presented in chapter 2, where the two presented systems perform a greedy search through the hypothesis space. This gives them a good exploitation power but a rather poor exploration power.

In order to overcome the limits and to exploit the good aspects of both approaches, in the next chapter we introduce a hybrid evolutionary system that tries to combine the good features of both approaches, in order to obtain a system that is good at both exploration and exploitation. This system uses a selection mechanism similar to the one adopted by REGAL and a high level representation inspired by SIA01. Only mutation operators are used for evolving individuals. In particular the mutation operators used can generalize and specialize a clause by means of some ILP operations, e.g., turning a constant into a variable. Moreover the particular mutation operators adopted do not act at random, but act greedily, by considering a number of mutation possibilities and applying the one yielding the best improvement in the fitness of the individual being mutated.

4.7 Conclusions

In this chapter we have seen examples of how EC can be applied to solve ILP problems. We first have motivated the choice of using EC at this aim. In particular we have given motivations on the use of GAs or GP, which have features that render them suitable for ILP tasks.

We have then presented five EAs for ILP. REGAL adopts a binary string representation, and genetic operators that can be used for generalizing or specializing clauses (by means of dropping or adding some conditions to the clause). REGAL introduces the US selection operator, which promotes species and niches formation without the need of using any distance measure. This operator is the base of the selection operators introduced in the next chapter. G-NET and DOGMA are highly based on REGAL, especially for what regards

the representation adopted. However they differ for the fitness functions they adopt, and in the case of G-NET also for the genetic operators used.

An high level representation is what characterizes SIA01. Clauses are represented by using a list of predicates, variables and constants. This representation allows the system to perform ILP aimed operations, e.g., turning a constant into a variable. This kind of operations are not directly possible to perform in systems with a representation like the one used by REGAL.

The last system presented, GLPS, adopts a Pittsburgh approach, by representing logic programs as a forest of AND-OR trees.

Chapter 5

Evolutionary Concept Learner

In this chapter we describe the hybrid evolutionary system for ILP ECL (for Evolutionary Concept Learner) (Divina and Marchiori, 2001; Divina and Marchiori, 2002; Divina, 2001b). ECL evolves a set of chromosomes representing clauses, where at each iteration fitter chromosomes are selected, mutated, and optimized. ECL follows the Michigan approach. Clauses are encoded in a high level language that allows the system to perform ILP aimed operations. In particular four mutation operators are used, two for generalizing clauses and two for specializing them.

One of the novelties with respect to previous approaches is the introduction of two stochastic mechanisms for controlling the complexity of the construction, optimization and evaluation of clauses. The first mechanism allows the user to specify the percentage of background knowledge that the algorithm will use, in this way controlling the computational cost of fitness evaluation. The second mechanism allows one to control the greediness of the operators used to mutate and optimize a clause, thus controlling the computational cost of the search. Furthermore ECL introduces a variant of the US selection operator, that helps the system in promoting diversity in the population while maintaining at the same time a good coverage of the positive examples. Finally, ECL is enhanced with methods for dynamic handling of numerical attributes. This last aspect is treated in chapter 6.

5.1 Motivations

We are interested in learning knowledge expressed in first–order formulas containing variables. In particular we are interested in knowledge represented by a fragment of first–order logic, called Horn clauses that do not contain negative literals nor function symbols. This knowledge can be directly used in programming languages based on logic programming, e.g., Prolog.

The approach used in the majority of first–order based learning systems, e.g., the systems presented in chapter 2, is to use specific search strategies, like

the general-to-specific (hill climbing) search (Quinlan, 1990) and the inverse resolution mechanism (Muggleton and Buntine, 1988). However, the greedy selection strategies adopted for reducing the computational effort render these techniques often incapable of escaping from local optima.

We have seen in chapter 4 that an alternative approach based on evolutionary computation can be used for inductive learning in FOL. This approach is motivated by two major characteristics of EAs, as we have seen in chapter 4: their good exploration power, that gives them the possibility of escaping from local optima, and their ability to cope well when there is interaction among arguments and when arguments are of different type. However the use of stochastic variation operators is often responsible for the rather poor performance of EAs on learning tasks which are easy to tackle by algorithms that use specific search strategies.

These observations suggest that the two approaches are applicable to partly complementary classes of learning problems. More important, they indicate that a system incorporating features from both approaches would benefit from the complementary qualities of the approaches. In fact, EAs are characterized by good exploration qualities, but by rather poor exploitation qualities. On the other hand standard ILP techniques have good exploitation quality but have less exploration power.

This motivates us to investigate a framework based on EAs for ILP that incorporates effective search strategies, like those used in ILP systems presented in chapter 2.

A common problem with ILP systems is the computational effort required for establishing the coverage of hypotheses, consequently, evaluation of individuals is the most costly operation in an EA for ILP. Therefore, we want to adopt some methods for improving the efficiency of evaluation of individuals. In particular, we intend to reduce the amount of time required by evaluation by a sampling of the background knowledge. This will result in using only a portion of the background knowledge available for a problem during the learning process.

Another feature that we want to incorporate in the system is a way for handling numerical values. The standard way for handling numerical values in ICL, is to discretize them into a number of intervals beforehand the learning process. During the learning process the obtained intervals are used instead of the numerical values. In this way numerical values can be treated as nominal. However this strategy is not always the best solution for dealing with numerical values. A system that adopts a simple technique for handling numerical values during the learning process is SIA01. In fact SIA01 can create and modify intervals for numerical values during the learning process. The bounds of the intervals are generated randomly, in a user given range. Other EAs for ILP, like those presented in chapter 4, do not include any method for dealing with numerical values. We believe that this is an important limitation. In fact ILP problems in which numerical values are present are common. In these cases, a good way for dealing with numerical values is essential for achieving good results.

5.2 The Learning Algorithm

The algorithm considers Horn clauses. Constants and variables are the only terms allowed to appear inside a clause, so functions can not appear inside a clause. ECL takes in input a set of positive examples, a set of negative examples and background knowledge and outputs a logic program describing the concept to be learned. Examples and background knowledge consist of a set of ground facts. Figure 5.1 shows an example of input taken by ECL if we want to learn the simple family concept of father. From the first positive example we know that $jack$ is the father of $bill$, and from the first negative example we know that eve is not the father of $bill$. The background knowledge for this problem consists of facts describing family relations between people and other features of people, such as the sex. For instance from the background knowledge we can find that $jack$ is a parent of $bill$, that eve is a woman, and so on. Information that is irrelevant to the learning task may as well appear in the background knowledge. For example in the background knowledge shown in the figure, it is stated that $tizio$ is married to $beth$. This information is useless for the task of learning the concept of father.

The concept of father can be described by a single clause of this form: $father(X, Y) \leftarrow parent(X, Y), male(X)$. In the following we will see how ECL can reach this solution starting from the input showed in figure 5.1.

Positive Examples	**Negative Examples**	**Background Knowledge**
father(jack,bill).	father(eve,bill).	parent(jack,bill).
father(giacobbe,noe).	father(eve,clint).	parent(jack,eve).
father(jack,eve).	father(matt,bill).	parent(eve,bill).
father(matt,jane).	father(matt,eve).	parent(matt,clint).
father(matt,luck).	father(jane,matt).	parent(matt,luck).
father(tizio,caio).	father(jane,luck).	parent(eve,clint).
father(tizio,sempronio).	father(caio,bill).	parent(tizio,caio).
father(lucio,cesare).	father(caio,eve).	parent(tizio,sempronio).
father(lucio,lucia).	father(caio,matt).	male(sempronio).
father(cesare,nerone).	father(sempronio,tizio).	married(tizio,beth).
father(cesare,cleopatra).	father(cesare,lucio).	relative(jack,bill).
	father(nerone,cesare).	male(jack).
	father(cleopatra,cesare).	male(bill).
	father(noe,cesare).	female(eve).
	father(giacobbe,sempronio).	female(cleopatra).
		male(clint).
		male(caio).
		male(tizio).

Figure 5.1: Example of input of ECL for the simple family relation concept of father.

Since the output of ECL can be directly used in Prolog, in the following we can also write a clause in Prolog syntax, where $\leftarrow$ is substituted with $:-$.

ECL works only with positive and negative examples, so it works with two classes of examples. When the examples belong to n classes, $n > 2$, then ECL is run on each class i, $1 \leq i \leq n$, using the ith class as positive examples, and the union of the other classes as negative examples. So a set of rules is learned for each class. To the rules induced, a classification procedure similar to the one of CN2 (Clark and Boswell, 1991) is applied: all rules whose conditions apply to a testing example are taken and the number of training examples of each class covered by the rules are summed up. The class with the largest sum is assigned to the testing example.

```
ALGORITHM(ECL)
 1  Sel = positive examples
 2  repeat
 3          Select partial BK
 4          Population = ∅
 5          while not terminate
 6          do Adjust weights of Sel
 7              Select n individuals using Sel
 8              for each selected individual φ
 9              do Mutate φ
10                  Optimize φ
11                  Insert φ in Population
12          Store Population in Final_Population
13          Sel = Sel - {positive examples covered by clauses in Population}
14    until max_iter is reached
15  Evaluate Final_Population using BK
16  Extract a solution from Final_Population
```

Figure 5.2: The overall learning algorithm ECL. In the selection a variant of the US selection operator is used.

Figure 5.2 gives the overall scheme of ECL and figure 5.3 gives a graphical scheme of ECL. In the **repeat** statement the algorithm constructs iteratively a `Final_Population` as the union of `max_iter` populations. The user can set the value of `max_iter` through the use of a parameter supplied to ECL before starting the learning process. At each iteration, part of the background knowledge is chosen using a stochastic search bias described below. A `Population` is evolved by the repeated application of selection, mutation and optimization (the **while** statement). These operators and the evaluation of individuals use only the chosen part of background knowledge. In chapter 2, we have seen how we can verify if a clause covers an example. In order to evaluate individuals, ECL poses a query for each example to be tested to the logic program formed by the union of the clause encoded by the individual and the partial background knowledge in use. Prolog is then used for verifying whether these queries have successful derivations or not. If a query relative to an example has

a successful derivation, then the example is covered by the clause, otherwise it is not.

At each generation n clauses are selected by means of a variant of the US selection operator, described in the following sections. The selected clauses are then modified using mutation and optimization operators, and are inserted in the population. The population is let grown in this way until a maximum size has been reached. When the maximum population size is reached, newly generated individuals will substitute some individuals of the current population. To this aim a tournament mechanism of size four is adopted. We adopt this strategy instead of replacing the worst individual in the population for efficiency reasons, in fact the population is not kept sorted. Four mutation operators are used. Two mutation operators can generalize clauses, while the other two can specialize clauses. The optimization phase consists of a repeated application of mutation operators to the selected individual.

When the construction of the `Final_population` is completed, a logic program is extracted using a procedure that builds a solution that is as accurate as possible.

In the following sections we address each feature of ECL in detail.

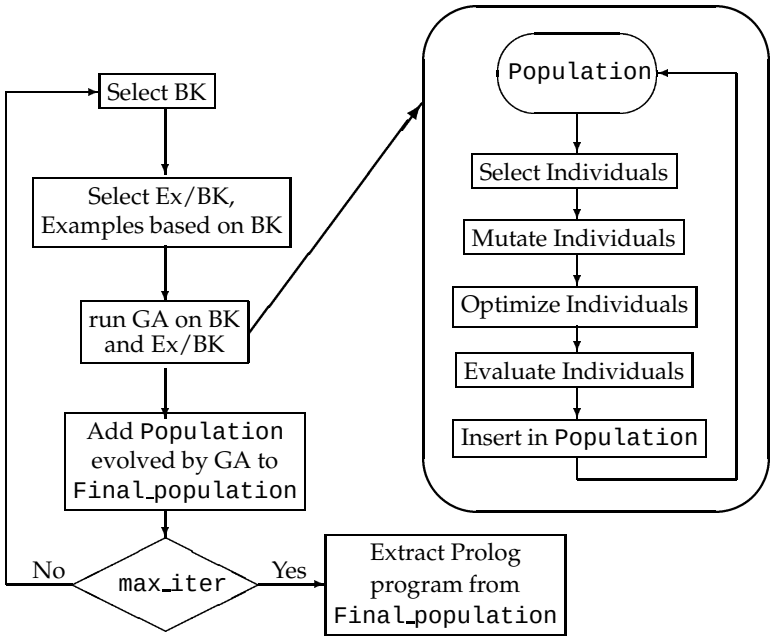


Figure 5.3: A schematic description of ECL.

5.3 Stochastic Search Biases

ECL uses two stochastic mechanisms, one for selecting part of the background knowledge in step 3 of figure 5.2, and one for selecting the degree of greediness of the operators used in the evolutionary process.

A parameter pbk (pbk real number in $(0, 1]$) is used in a simple stochastic sampling mechanism which selects an element of the background knowledge with probability pbk. In this way the user can limit the cost of the search and fitness evaluation by setting pbk to a low value. This because only a part of the background knowledge will be used when assessing the goodness of an individual. This leads to the implicit selection of a subset of the examples (only those examples that can be covered with the partial background knowledge selected will be considered). Individuals will be evaluated on these examples using only the partial background knowledge. In this way an individual can be wrongly evaluated because a subset of the examples is used, and also because those examples can be wrongly classified, in case they are covered using the whole background knowledge, but are not covered using the partial background knowledge.

Example 5.1 Suppose we have selected from the background knowledge of figure 5.1 only the following five facts: $parent(matt, luck)$, $parent(jack, bill)$, $male(jack)$, $male(bill)$, $female(eve)$. Let $\varphi = father(jack, X) \leftarrow parent(jack, X), male(jack)$ be the individual that has to be evaluated. Then the positive example $father(jack, bill)$ will be correctly classified, while the positive example $father(jack, eve)$ will be wrongly classified as negative. However if the whole background knowledge were used then both the examples would be correctly classified by φ.

This is different from other mechanisms used for improving the efficiency of fitness evaluation, like (Glover and Sharpe, 1999; Teller and Andre, 1997; Sebag and Rouveirol, 1997), where training set sampling is employed for speeding up the evaluation of individuals.

The construction, mutation and optimization of a clause uses four greedy generalization/specialization operators (described later in an apart section). Each greedy operator involves the selection of a set of constants, variables or atoms, depending on the particular mutation operator. The size of this set can be supplied by the user by setting a corresponding parameter Ni ($i = 1, \ldots, 4$). The elements of the set are then randomly chosen with uniform probability. Each element of the set represents a mutation possibility. Each possibility is tested, and the one yielding the best improvement in the fitness of the individual is applied. In this way the user can control the greediness of the operators, where higher values of the parameters imply higher greediness.

Finally ECL uses also a language bias which is commonly employed in ILP systems for limiting explicitly the maximum length of clauses.

These search biases allow one to reduce the cost of both the search and fitness evaluation, but the price to pay may be the impossibility of finding the

best clauses.

5.4 Fitness Function and Encoding

The fitness function assigned to an individual is the inverse of its accuracy. So the fitness of an individual φ will be:

$$f(\varphi) = \frac{1}{Acc(\varphi)} = \frac{|E^+| + |E^-|}{p_\varphi + (|E^-| - n_\varphi)} \tag{5.1}$$

The aim of ECL is to minimize the fitness.

ECL uses a high level representation similar to the one used by SIA01 (see section 4.4), where a rule $p(X, Y) \leftarrow r(X, Z), q(Y, a).$ can be described by an individual of the following form:

$$\boxed{p, X, Y}\ ,\boxed{r, X, Z}\ ,\boxed{q, Y, a}\ ,(t,Z,b)$$

where the framed atoms are active while the atom between brackets is not. Atoms that are not active are not considered when the clause is evaluated. However they can be activated during the evolutionary process. In this way it is easy to represent rules of variable length, and to access and process atoms contained in the rule. The length of a clause C, $length(C)$, is defined as the number of active atoms in C. For instance the length of the above clause is three.

5.5 Selection Operator

In step 7 of the algorithm shown in figure 5.2, individuals can be selected by three selection operators: the standard US selection operator or two variants of it. The standard US selection operator is described in chapter 4. The first variant of the US selection operator is called Weighted US (WUS) selection operator (Divina and Marchiori, 2002), while the second is called Exponentially Weighted US (EWUS) selection operator (Divina et al., 2002; Divina and Marchiori, 2004b). The second variant is used by default.

5.5.1 Why the Two Variants of the US Selection Operator?

The US selection operator has various good characteristics. With the US selection operator individuals with a high coverage and with good fitness are favored. This is because individuals with a high coverage participate in more roulette wheels, and individuals with high fitness have wider sectors in the roulette wheel tournaments in which they participate. In this way a good coverage of positive examples is promoted. Speciation is obtained through the use of the US selection operator. In fact only individuals covering the same examples compete with each other for being selected.

The selection operator is called "universal suffrage" because positive examples can be viewed as voters and chromosomes as candidates to be elected. All examples have the same right (probability) of voting. The US operator does not distinguish between examples that are harder to cover, and this can favor the emergence of so called super–individuals that cover many (easy) examples. Super–individuals will be often selected, and this may lead to a population characterized by a low diversity. This phenomenon can negatively affect the solution, since there are less clauses to choose from, in order to build the final solution.

In REGAL this problem is tackled by using the distributed architecture of nodes where each node evolves a single population acting on a different set of examples, and the nodes co-evolve in order to favor the formation of different species. A supervisor process can shift the focus of the genetic search performed by each genetic node that constitute the system by simply assigning a different set of training examples to the nodes. In this way the system promotes diversity. However this strategy is not adopted in ECL, because ECL was not born as a parallel system. For this reason the two following operators are introduced.

5.5.2 WUS Selection Operator

The first variation of the US selection operator is represented by the WUS selection operator. The difference between this operator and the standard US selection operator lies in the first step of the selection. In the WUS operator examples do not have the same voting power, i.e., examples are no longer chosen at random. A weight is associated to each example, where smaller weights are associated to examples harder to cover. More precisely the weight for an example e_i is equal to:

$$w_i = \frac{|Cov(e_i)|}{|Pop|} \tag{5.2}$$

$1 \leq i \leq P$, being $|Pop|$ the population size and P the number of positive examples. If the population is empty, then each example has the same weight. Examples are now chosen with a roulette wheel mechanism, where the sector associated to each example is inversely proportional to its weight. So the fewer individuals cover an example, the more chances that example has of being selected in the first step of the selection.

5.5.3 EWUS Selection Operator

The WUS selection operator selects with higher probability examples that are harder to cover, by means of the weights assigned to each example. The EWUS selection operator continues this line, with the difference that now the weight

w_i of an example e_i, $1 \leq i \leq P$, is given by the following formula:

$$w_i = \frac{e^{-|Cov(e_i)|}}{\sum_{j=1}^{P} e^{-|Cov(e_j)|}} \tag{5.3}$$

Examples are still selected with a roulette wheel mechanism, where the sector associated to each example is proportional to its weight. In this way the selection pressure toward examples harder to cover will be much higher than in the WUS selection operator.

In both the WUS and the EWUS operators, the second step of the selection is the same as the one performed by the US operator. In both the WUS and the EWUS selection operators, at the beginning of each generation the weights assigned to the positive examples are updated (step 6 of the algorithm of figure 5.2).

Example 5.2 Suppose we have three positive examples e_1, e_2, e_3, and five individuals: $\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5$. Let $Cov(e_1) = \{\varphi_1, \varphi_2\}$, $Cov(e_2) = \{\varphi_2, \varphi_3, \varphi_4\}$, $Cov(e_3) = \{\varphi_5\}$. Then each example has the same probability of being selected in the first step of the US selection operator. The weights assigned to each example by the WUS and the EWUS operators are those reported in table 5.1.

Operator	$w(e_1)$	$w(e_2)$	$w(e_3)$
WUS	0.4	0.6	0.2
EWUS	0.2477	0.0901	0.6653

Table 5.1: Weights assigned to each example by the WUS and the EWUS operator.

The selection probability relative to each example e_i, $1 \leq i \leq 3$, in the WUS operator is inversely proportional to the weight assigned to e_i. The difference between the WUS and the EWUS operators is that in the former operator the weights are linear, while in the latter the weights are exponential. For this reason, e_2 has a small chance of being selected by the EWUS operator, while the chance that e_2 has of being selected by the WUS operator is higher. Another observation is that e_3 is the example that will likely be selected by the EWUS operator. Also in the WUS operator e_3 has a probability of being selected higher than the other examples. However this probability is much smaller in the WUS operator than in the EWUS operator.

5.5.4 Discussion on Selection

With the introduction of the two weighted variants of the US selection operator we wanted to achieve the following aims:

1. Good coverage of positive examples.
2. Diversity in the population.

One would usually like to have the final hypothesis found by the GA to cover as many positive examples as possible and as few negative examples as possible. The fitness function deals with the coverage of positive examples. The variants of the US selection operator deal also with the second aspect. By having the learning system focusing more and more on difficult examples to cover, it is easier to have each positive example covered by at least one individual. It is also easier to have more diversity in the population.

Maintaining a good diversity in the population would also lead to a good coverage of examples. Generally it is a good idea to have the population spread across the hypothesis space, so that all the areas of the hypothesis space can be searched. We have seen some methods for maintaining diversity in section 3.5. Those methods implied the use of some distance measure in order to establish the similarity between individuals. Establishing a distance measure is not an easy task when the rules are expressed in FOL. The US selection operator promotes species and niches formation without the need of any distance measure.

The US selection operator considers all the examples as equally important. In this way the search can focus mainly in the areas where the concentration of hypotheses covering many (easy) positive examples is higher, and seldom touches less inhabited areas of the hypothesis space, because hypotheses in these areas may cover few (difficult) examples. With the weighted variants of the US operator, examples difficult to cover will have higher chance of being selected. In this way, if we have many individuals covering the same subset of training examples, those individuals will be seldom selected for reproduction, because that area of the hypothesis space is already crowded. Instead we will select, or create, individuals that occupy, or will occupy, less exploited regions.

Formally, the probability that an individual φ has of being selected can be written as:

$$P(\varphi) = \sum_{e_i \in p_\varphi} P(\varphi|e_i) \cdot P(e_i)$$

where $P(\varphi|e_i)$, the probability of φ being selected conditioned to the selection of example e_i in p_φ, is equal to

$$\frac{f(\varphi)}{\sum_{\varphi \in Cov(e_i)} f(\varphi)}$$

with $f(\varphi)$ the fitness of individual φ. The three US based selection operators induce different probability $P(e_i)$ of selection of example e_i.

In the US selection operator $P(e_i) = \frac{1}{|E^+|}$. No distinction is made among positive examples, so individuals with a high *recall* are favored in this scheme, because they cover many positive examples. The recall of an individual is defined as follows:

Definition 5.1 Let φ be an individual. Then the recall of φ is defined as $\frac{p_\varphi}{|E^+|}$.

A possible effect of favoring individuals with high recall, could be the presence of super individuals, that will dominate the population and will be often selected, leading to a low diversity in the population. Also some examples could stay uncovered, because not selected and difficult to cover.

In the WUS selection operator $P(e_i)$ is proportional to $\frac{1}{w_i+1}$, with w_i defined in equation 5.2. In this way WUS tries to favor not only examples that are uncovered, but in general it favors examples that are difficult to cover. Individuals with a high recall will still have more chances of being selected. However the system will focus more and more on examples harder to cover. In this way it is more probable that at the end of the evolutionary process all positive examples are covered. The weights given to examples by the WUS operator increase linearly with each individual that covers the example. This means that the chances that an example covered by one individual is selected are almost the same as the ones of an uncovered example. The EWUS selection operator deals with this problem, by choosing $P(e_i) = w_i$, with w_i defined in equation 5.3, which changes exponentially. If there are uncovered examples, then the probabilities that they have of being selected with the EWUS operator are very high, and also examples covered by few individuals are likely to be chosen.

In this way a good coverage of positive examples is encouraged, leading also to a good diversity in the population.

Individuals with a high recall will be selected more often only if there are not multiple copies of them, because in this case the examples that they cover will have an high coverage rate, and if these individuals do not cover only easy examples.

Example 5.3 Suppose the population is formed by four individuals φ_i, $1 \leq i \leq 4$, and that the clause encoded by φ_1 is the same as the one encoded by φ_2, while the other two individuals encode different clauses. Suppose there are five positive examples e_j, $1 \leq j \leq 5$ and that the coverage sets are the following:

$$\begin{aligned} Cov(e_1) &= \{\varphi_1, \varphi_2\} \\ Cov(e_2) &= \{\varphi_1, \varphi_2, \varphi_3\} \\ Cov(e_3) &= \{\varphi_3\} \\ Cov(e_4) &= \{\varphi_3\} \\ Cov(e_5) &= \{\varphi_4\} \end{aligned}$$

In this case φ_1 and φ_2 have less chances of being selected in the WUS and EWUS selection operators than the other two individuals, because even if their recall is good they cover only "easy" examples. On the other hand φ_3 is the individual that has more probability of being selected, since it has a good recall and it covers also examples that are not "easy".

5.6 Clause Construction

A clause C is constructed when the selection operator selects a positive example which is not yet covered by any clause in the actual population. This example is used as seed in the following procedure, where BK_{pbk} denotes the part of background knowledge selected at step 3 of ECL.

1. The selected example becomes the head of the emerging clause;
2. Construct two sets A_C and B_C. A_C consists of all atoms in BK_{pbk} having all arguments appearing in the head; B_C contains all elements in $BK_{pbk} \setminus A_C$ having at least one argument occurring in the head.
3. **while** $length(C) < l$ and $A_C \cup B_C \neq \emptyset$
   (a) Randomly select an atom from A_C and remove it from A_C. If A_C is empty then randomly select an atom from B_C (and remove it from B_C). Add the selected atom to the body of the emerging clause C.
4. Generalize C as much as possible by means of the repeated application of the generalization operator "constant into variable" (described in section 5.7). Apply this operator to the clause until either its fitness increases or a maximum number of iterations is reached. In the former case, retract the last application of the generalization operator.

In step 3, l is the maximum length of a clause, supplied by the user. If l was not supplied then the first condition of the *while* cycle is dropped, and no constraint on the length of the clause is imposed.

In the second step, there is also the possibility to consider a third set B'_C, consisting of all the atoms of BK_{pbk} having at least one argument appearing in $A_C \cup B_C$. When B'_C is constructed B_C becomes $B_C \cup B'_C$.

At the end of the procedure, if $|A_C| > l$, the atoms of A_C that were not added to C are moved into B_C. Thus B_C contains all the atoms that are not considered to be in the body of the C, called non–active atoms.

Example 5.4 Suppose the example $e = father(jack, bill)$ from figure 5.1 is selected, and let $Cov(e) = \emptyset$. Then a new clause C covering e must be created. Suppose that we use all the background knowledge shown in figure 5.1, and let $l = 4$. From the first step we have that $C = father(jack, bill) \leftarrow$. From the second step we have that A_C is $\{parent(jack, bill), relative(jack, bill)\}$, and $B_C = \{parent(jack, eve)\ , parent(eve, bill)\ , male(jack)\ , male(bill)\}$.

Now we start adding atoms to the body of C. We add $parent(jack, bill)$ and $relative(jack, bill)$, since they are the only atoms present in A_C. A_C is now equal to $\emptyset$, so we select an atom from B_C, let it be $male(jack)$. We then add it to C and remove it from B_C.

Now $length(C) = 4 = l$ so the final result of this initialization step is $father(jack, bill) \leftarrow parent(jack, bill), relative(jack, bill), male(jack)$.. We still need to perform step 4 of the initialization procedure. We can generalize the

clause by applying $\{jack/X\}$, and $\{bill/Y\}$. No further generalizations are possible, and the final result is already the target concept: $father(X, Y) \leftarrow parent(X, Y), relative(X, Y), male(X)$..

5.7 Mutation and Optimization

For moving in the hypothesis space ECL makes use of four mutation operators, and takes advantage of the general-to-specific order of the hypothesis space. In fact, two mutation operators are used for generalizing clauses, and two for specializing. Here we use the concept of generality as defined in section 2.2. A clause can be then generalized by either deleting an atom from its body (the atom is deactivated) or by turning a constant into a variable. With the inverse operations a clause can be specialized.

These mutation operators do not act completely at random, but consider a number of mutation possibilities and among these possibilities the best one is applied. The number of mutation possibilities considered are determined by the values of four parameters $N_1, \ldots, N_4$. Table 5.2 shows for each mutation operator the associated parameter N_i, $1 \leq i \leq 4$.

Mutation Operator	Associated N_i
Atom Deletion	N_1
Constant into Variable	N_2
Atom Addition	N_3
Variable into Constant	N_4

Table 5.2: Mutation operators with associated parameters for controlling the greediness.

In order to determine which mutation possibility is the best, a gain function is used. When applied to clause C and mutation operator τ, the gain function yields the difference between the clause fitness before and after the application of τ:

$$gain(C, \tau) = f(C) - f(\tau(C))$$

In the following with $Cov_\varphi = [p_\varphi/n_\varphi]$ we indicate the number p_φ of positive and n_φ of negative examples covered by an individual φ. The four operators are defined below.

Atom Deletion

Consider the set Atm of N_1 atoms of the body of C randomly chosen. If the number of atoms in the body of C is less than N_1 then Atm contains all the atoms in the body of C. For each A in Atm, compute $gain(C, -A)$, the gain of C when A is deleted from C.

Choose an atom A yielding the highest gain $gain(C, -A)$ (ties are randomly broken), and generalize C by deleting A from its body.

Insert the deleted atom A in a list D_C associated with C containing atoms which are deactivated. Atoms from this list may be added to the clause (activated) during the evolutionary process by means of a specialization operator.

Example 5.5 Consider the problem of learning the concept of father from the input given in figure 5.1. Suppose that the parameter N_1 is set to 2, and that an individual φ has been selected. Let, moreover, φ encode the following clause:

$$father(X, bill) :- parent(X, bill), male(X), parent(tizio, bill), male(bill).$$

For the selected individual $Cov_\varphi = [0/0]$. Then an application of the *atom-deletion* operator could work as follows. Two atoms are randomly chosen for being considered for deletion. Let them be $male(X)$ and $parent(tizio, bill)$. Deleting the atom $male(X)$ does not modify Cov_φ, while without the atom $parent(tizio, bill)$ Cov_φ becomes $[1/0]$ yielding to a better fitness of φ. So the atom $parent(tizio, bill)$ is deleted from the body of the clause.

Constant into Variable

Consider the set Var of variables present in C plus a new variable. Consider also the set Con consisting of N_2 constants of C randomly chosen. If the number of constants of C is less than N_2 then Con consists of all the constants of C.

For each a in Con and each X in Var, compute $gain(C, \{a/X\})$, the gain of C when all occurrences of a are replaced by X.

Choose a substitution $\{a/X\}$ yielding the highest gain (ties are randomly broken), and generalize C by applying $\{a/X\}$.

Example 5.6 Suppose we select the clause

$$father(X, bill) :- parent(X, bill), male(X), male(bill).$$

encoded by an individual φ and suppose that the parameter N_2 is set to 1. Then $Var = \{X, Y\}$ and $Con = \{bill\}$. The two substitution $\theta_1 = \{bill/X\}$ and $\theta_2 = \{bill/Y\}$ are applied to the clause. With the application of θ_1, Cov_φ becomes $[0/0]$. With the application of θ_2, Cov_φ becomes $[3/0]$ improving in this way the fitness. So θ_2 is applied to the clause.

Atom Addition

Consider the set Atm consisting of N_3 atoms of B_C (list built at initialization time) and of N_3 atoms of D_C, all randomly chosen. If B_C contains less than N_3 atoms, then Atm contains all the atoms of B_C. The same holds for D_C.

For each A in Atm compute $gain(C, +A)$, the gain of C when A is added to the body of C.

Choose an atom A yielding the highest gain $gain(C, +A)$ (ties are randomly broken), and specialize C by adding A to its body.

Remove A from its original list (B_C or D_C).

Example 5.7 Suppose the individual φ represents the clause

$$father(X, bill) : -parent(X, bill), male(bill).$$

has been selected, and suppose that N_3 is set to 2. $Cov_\varphi = [1/1]$. The *atom-addition* operator could consider to add the atoms $male(X)$ and $female(X)$ to the body of the clause. The addition of $male(X)$ changes Cov_φ to $[1/0]$, while the addition of $female(X)$ changes Cov_φ to $[0/1]$. So the atom $male(X)$ is added to the body of the clause.

Variable into Constant

Consider the set Con consisting of N_4 constants (of the problem language) randomly chosen, and a variable X of C randomly chosen. If the constants of the problem language are less than N_4, then Con contains all the available constants.

For each a in Con, compute $gain(C, \{X/a\})$, the gain of C when all occurrences of X are replaced by a.

Choose a substitution $\{X/a\}$ yielding the highest gain (ties are randomly broken), and specialize C by replacing all occurrences of X with a.

Example 5.8 Suppose the individual φ representing the clause

$$father(X, Y) : -parent(X, Y), male(Y), female(X).$$

has been selected, and suppose that N_4 is set to 2. $Cov_\varphi = [0/2]$. Let $Con = \{jack, eve\}$ and X the variable chosen. Then the operator will apply the substitutions $\theta_1 = \{X/jack\}$ and $\theta_2 = \{X/eve\}$. The application of θ_1 yields to $Cov_\varphi = [0/0]$, while the application of θ_2 yield to $Cov_\varphi = [0/2]$. So θ_1 is applied to the clause.

The optimization phase, performed after an individual is mutated, is shown in figure 5.4. Optimization consists of a repeated application of the greedy operators to the selected individual, until either its fitness does not increase or a maximum number of iterations (in figure denoted as max_opt_steps) is reached. The default value of max_opt_steps is 10, and can be modified by setting a relative parameter. If the procedure ends because an application of a mutation operator negatively affected the fitness of the individual being optimized then the last mutation applied is retracted. This control, and the relative action, is performed in the `if` statement in step 7 of the procedure.

ECL does not use any crossover operator. Experiments with a simple crossover operator, which uniformly swaps atoms of the body of the two clauses, have been conducted. However the results did not justify its use. Other EAs

```
OPTIMIZATION( φ, max_opt_steps )
 1  opt_steps = 1, cont = true
 2  while (opt_steps < max_opt_steps ) ∧ (cont)
 3  do
 4       φ' = mutate(φ)
 5       evaluate(φ')
 6       opt_steps = opt_steps + 1
 7       if fitness(φ) < fitness(φ')
 8         then cont = false
 9         else cont = true
10              φ = φ'
11  return φ
```

Figure 5.4: Optimization function used in step 9 of the algorithm of figure 5.2. The procedure takes as input an individual φ and a maximum number of optimization steps, max_opt_steps.

do not use any crossover operators. For instance EP and ES usually rely only on mutation for carrying out the evolutionary process.

5.8 Hypothesis Extraction

The termination condition of the main while statement of ECL, in step 5 of figure 5.2, is met when either all positive examples are covered or a maximum number of iterations is reached. In this case a logic program for the target predicate is extracted from the final population.

In a early version of ECL the problem of extracting a solution from the final population was translated into an instance of the weighted set covering problem as follow. Each element φ of the final population was a column with positive weight equal to

$$weight_{\varphi} = f(\varphi) + 1 \tag{5.4}$$

Each covered positive and uncovered negative example was a row. The columns covering each positive example were the clauses that covered that example. The columns relative to each uncovered negative examples were the clauses that did not cover that negative example. In this way clauses covering few negative examples were favored. The task was to find a set of columns (clauses) with minimal total weight that cover all the rows (hence the positive examples). A fast heuristic algorithm (Marchiori and Steenbeek, 2000) was applied to this problem instance to find a "best" theory. However this approach turned out not to be the best one for this problem, as illustrated in example 5.9.

In (Divina and Marchiori, 2002), the problem was then translated differently

having weights equal to

$$weight_\varphi = n_\varphi + 1 \tag{5.5}$$

and each covered positive example was a row. Again the columns relative to each positive example were the clauses that covered that example. However also this solution had problems in extracting hypotheses that excluded negative examples. This is due to the fact that the procedure considers the weight of the emerging solution only as the sum of the weights of the individuals in the solution, without considering possible overlapping among negative examples covered.

Positive	$Cov(p_i)$				Negative	$Cov(n_i)$			
p_1	φ_1			φ_4	n_1	φ_1		φ_3	
p_2	φ_1	φ_2	φ_3	φ_4	n_2	φ_1		φ_3	
p_3	φ_1	φ_2	φ_3	φ_4	n_3		φ_2		
p_4	φ_1		φ_3	φ_4	n_4		φ_2		φ_4
p_5			φ_3	φ_4	n_5				φ_4
p_6			φ_3	φ_4	n_6				φ_4

Table 5.3: Coverage of positive and negative examples for example 5.9.

Individual	weight 5.4	weight 5.5
φ_1	2.50	3
φ_2	3	3
φ_3	2.33	3
φ_4	2.33	4

Table 5.4: Weights assigned to each individual by the procedures for extracting the final solution using weights defined in equations 5.4 and 5.5.

Example 5.9 Suppose we have six positive examples e_i, $1 \leq i \leq 6$ and six negative examples. Suppose that in the final population there are four individuals that cover the training examples as in table 5.3.

The weights assigned to each individual by the procedures using the weights given by equation 5.4 and 5.5 are given in table 5.4.

Both the procedures based on the weighted set covering algorithm extract a solution formed by only φ_4, since this solution covers all positive examples with minimal total weight. The total weight of this solution is 2.33 in case of weights given by equation 5.4, and 4 in case of weights given by equation 5.5. The accuracy of this solution is $\frac{6+3}{12} = \frac{9}{12}$. In this case a solution formed by φ_1 and φ_3 has better accuracy, equal to $\frac{6+4}{12} = \frac{10}{12}$. In both the procedures

based on the weighted set covering algorithm, the solution formed by φ_1 and φ_3 has a higher weight than φ_4. In fact the procedure using weights given by equation 5.4 assigns a total weight equal to 4.83 to this solution, while the weight assigned to $\{\varphi_1, \varphi_3\}$ by the procedure using weights given by equation 5.5 is 6.

```
EXTRACT SOLUTION(Final_Population)
 1  H = ∅, Pop = Final_Population, cont = true
 2  while Pop and cont
 3  do Let φ the most precise individual in Pop
 4     if p_φ ≠ ∅
 5       then H = H ∪ φ
 6            Pop = Pop \φ
 7            if Accuracy(H) < Accuracy({H − φ})
 8              then H = H \ φ
 9                   cont = false
10              else  Extract p_φ from E+
11                    Recompute the precision of all φ' ∈ Pop using E+
12       else cont = false
13  return H
```

Figure 5.5: Procedure for hypothesis extraction adopted by ECL.

A different approach to the problem was then experimented successfully. A new procedure for extracting the solution was adopted. This procedure is very simple, and illustrated in figure 5.5.

In the figure *precision* is defined as follow.

Definition 5.2 Let φ be an individual. Then the precision of φ is defined as $\frac{p_\varphi}{p_\varphi + n_\varphi}$.

Precision measures the proportion of examples that are correctly classified, while accuracy measures how well H classifies the training examples. Another measure that can be used for measuring the goodness of a hypothesis is recall (definition 5.1), which measures the proportion of positive examples that are correctly classified.

In step 3 the most precise individual in the population is selected. If two individuals have the same precision then the one covering more positive examples is chosen. If both individuals cover the same number of positive examples then one is randomly chosen. In step 6 the selected individual is extracted from the population. If the accuracy of the solution with the new added individual has dropped (step 7), then the individual is removed from the solution and the procedure stops. If that is not the case, in step 10 all the positive examples covered by the selected individual are extracted from the training examples set.

In step 11 the precision of all the remaining individuals in the population is recomputed, as now there are less positive examples.

We have seen that a first stopping criterion is represented by the accuracy of the emerging solution. As far as it does not decrease the procedure can be repeated. Other stopping criterion are that the set of positive examples covered by the extracted individual must not be empty (step 4), and that there need to be individuals in the population, otherwise it means that all the population has been added to the emerging solution. With this procedure the first clauses inserted in the solution are the most precise ones. Mistakes in classifying negative examples as positive are therefore unlikely to happen.

p_i	$Cov(p_i)$	n_i	$Cov(n_i)$
p_1	φ_1	n_1	φ_3
p_2	φ_1	n_2	
p_3	φ_1	n_3	
p_4	φ_2	n_4	
p_5	φ_2, φ_3	n_5	φ_1

Table 5.5: $Cov(p_i)$ and $Cov(n_i)$ for the example 5.10.

Example 5.10 Suppose we have five positive examples, p_i, $1 \leq i \leq 5$ and five negative examples, n_i. Let `Final_Population` be made of three individuals, $\varphi_1, \varphi_2, \varphi_3$. Let $Cov(p_i)$ and $Cov(e_i)$ be those given in table 5.5. The precisions for the three individuals are $\frac{3}{4}$, 1, $\frac{1}{2}$,respectively. At the first iteration of the "extract solution" procedure, φ_2 is chosen in the third step and is added to H. It is then removed from Pop, and the positive examples covered by φ_2 are removed from E^+. The recomputed precision (step 11) for φ_1 is still $\frac{3}{4}$, and for φ_3 is 0, since now $p_{\varphi_3} = \emptyset$. The accuracy of H is $\frac{2}{10}$. At the next iteration φ_1 is the most precise individual and is added to H. Now all the positive examples are removed. The accuracy of H is $\frac{9}{10}$, so the procedure it iterated again. φ_3 is selected, but since $p_{\varphi_3} = \emptyset$, the procedure ends and $H = \{\varphi_2, \varphi_1\}$ is returned as the final solution.

Example 5.11 Consider the population and examples given by example 5.9. The procedure based on precision extracts a solution composed by φ_3 and φ_1, in this order. This solution has an accuracy of $\frac{6+4}{12} = \frac{10}{12}$.

In chapter 7 we experimentally verify that the procedure based on precision is able to extract solutions of better quality than the solutions extracted by the other two procedures based on the weighted set covering algorithm.

Algorithm	Encoding	Approach	Fitness features
ECL	high level representation	Michigan	Inverse accuracy computed on partial BK

Table 5.6: Encoding, approach and fitness adopted by ECL.

Algorithm	Type of crossover	Mutation	Selection
ECL	none	2 greedy generalizing 2 greedy specializing optimization	US, WUS, EWUS

Table 5.7: A summary of the various operators adopted by ECL.

5.9 Conclusions

In this chapter we have presented the hybrid evolutionary system ECL for ILP. We have described in detail the various components of the systems. The most characterizing features of ECL are the use of greedy mutation operators, the use of a high level representation, the use of a stochastic sampling of the background knowledge and the incorporation of an optimization phase that follows the mutation. Tables 5.6 and 5.7 summarize the features of ECL.

The high level representation allows ECL to perform some ILP oriented operations, e.g., turning a particular variable into a particular constant. Moreover it allows the implementation of greedy ILP oriented mutation operators. When a mutation operator is applied to an individual, a number of mutation possibilities are tested, and the one yielding the best improvement in the fitness of the individual is then applied. The number of possibilities considered determines the greediness of each operator. The greediness of each mutation operator can be tuned by setting a relative parameter. Different values of the parameters determine different search strategies: low values yield weak learning methods, like standard EAs, while high values yield more greedy learning methods, like ILP systems, e.g., FOIL.

A high level representation has the advantage over a binary string representation of a direct way to define operators for generalizing and specializing a rule, and allows a more flexible form of the rules. Another advantage of an high level representation is that with such a representation it is easy to have individuals of variable length. With a binary representation, some problems can arise when dealing with numerical values. In fact the binary representation of the model can become quite long and this may slow down the process. Another advantage of the adopted representation, is that the user is not required to provide any model of the rules to be learned, as it happens, e.g., in REGAL. In ECL it is the example used as seed that determines the model of the rule. Furthermore, different examples determine different models for the rules, so

that the model is not an instance of a pre-defined scheme, as in REGAL.

The default selection operator, the EWUS selection operator, adopted by ECL promotes diversity in the population, as well as a good coverage of positive examples. An experimental evaluation of the US, the WUS and the EWUS selection operator is performed in chapter 7.

At the end of the evolutionary process a solution has to be extracted from the final population. To this aim, a simple procedure is adopted. The procedure first detects the most precise individual in the population, and adds it to the emerging solution. If the accuracy of the solution has not decreased then the procedure is repeated. This procedure solved some problems that were present in previous solutions adopted for this task.

In section 5.1, we stated that we wanted to introduce some methods for dealing with numerical values. The way ECL deals with numerical values is the subject of the next chapter.

Chapter 6

Treating Numerical Values

Real life learning tasks are often described by nominal as well as continuous, real-valued, attributes. However, most inductive learning systems treat all attributes as nominal, hence cannot exploit the linear order of real values. This limitation may have a negative effect not only on the execution speed but also on the learning capabilities of such systems.

In order to overcome these drawbacks, continuous-valued attributes are transformed into nominal ones by splitting the range of the values of the attribute in a finite number of intervals. Alternatively, continuous attributes are treated by means of inequalities, describing attribute subranges, whose boundaries are computed during the learning process. This process, called discretization, is *supervised* when it uses the class labels of examples, and *unsupervised* otherwise. Discretization can be applied prior or during the learning process (*global* and *local* discretization, respectively), and can either discretize one attribute at a time (*univariate* discretization) or take into account attributes interdependencies (*multivariate* discretization) (Dougherty et al., 1995).

Researchers in the Machine Learning community have introduced many discretization algorithms. An overview of various types of discretization algorithms can be found, e.g., in (Freitas, 2002; Freitas and Lavington, 1998; Kohavi and Sahami, 1996; Liu et al., 2002). Most of these algorithms perform an iterative greedy heuristic search in the space of candidate discretizations, using different types of scoring functions for evaluating a discretization. For instance, the popular Fayyad & Irani's discretization algorithm (Catlett, 1991; Fayyad and Irani, 1993) considers one attribute at a time, uses an information class entropy measure for choosing a cut point yielding a partition of the attribute domain, applies recursively the procedure to both the partitions, and uses the minimum description length as criterion for stopping the recursion.

In this chapter we first illustrate through an example a weak point of univariate discretization, which does not take into consideration the possible interactions among attributes. This motivates the introduction of multivariate discretization methods. A popular univariate discretization method is described. This method is a global supervised recursive algorithm, that uses the class en-

tropy information for discretizing numerical attributes.

We then present two methods for dealing with numerical attributes, which have been incorporated in ECL. The first method is a local unsupervised discretization method, here called ECL-LUD and was first introduced in (Divina et al., 2003c). The other method was introduced in (Divina et al., 2003b; Divina and Marchiori, 2004a) and is a local supervised method, of which two variants are proposed. In chapter 7 we will experimentally compare the proposed methods. Both methods treat numerical attributes by means of inequalities. An inequality is introduced in a rule each time a continuous attribute value of an example is considered. Inequalities are then modified during the evolution of rules. The way these inequalities are initialized and modified is different in the two methods.

Differently from the propositional case, where an attribute has exactly one value for each example, in ILP a numerical argument can have more values per example.

Example 6.1 Consider the molecule shown in figure 2.1. Suppose that that molecule is an example of a learning problem. The molecule is formed by a variable number of atoms. Each atom of each example is described by a number of proprieties, e.g., its charge. In this case an example will have more values for the argument charge, one for each atom composing the molecule.

This can negatively influence methods based on class entropy. An example with many values for a numerical argument has a high impact in the calculation of the average class entropy. In order to overcome this problem we propose three methods for dealing with numerical values. Information about class entropy is used only in the initialization of inequalities. However inequalities can be modified during the learning process, and so erroneous initializations of inequalities can be corrected.

In section 6.6 we give an overview of some other methods for dealing with numerical values in ICL, and in section 6.7 we summarize the content of this chapter.

6.1 Weak Point of Univariate Discretization

A typical example showing a drawback of univariate discretization methods based on class information entropy is the problem of separating the two classes shown in the figure 6.1, where positive and negative examples are labeled $+$ and $\times$. In this problem, every example is described by two attributes $attr_1$ and $attr_2$ uniformly distributed on the interval [0,1000]. The examples are equally divided in two classes. The solution to this learning problem is represented by the two lines in figure 6.1, described by the two rules $(attr_1 \leq 500) \wedge (attr_2 \leq 500)$ and $(attr_1 > 500) \wedge (attr_2 > 500)$.

Univariate discretization methods based on class information entropy, are not able to capture the interdependencies among attributes. For this reason

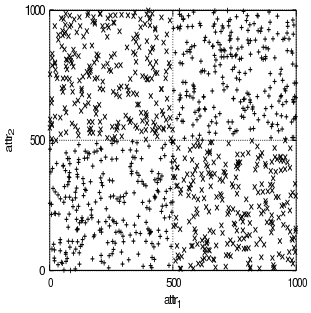


Figure 6.1: An artificially generated dataset, for which univariate global discretization is unlikely to work.

globally discretizing numerical attributes with such methods, e.g., with the Fayyad & Irani's algorithm, yields the risk of missing the necessary information to find the correct solution. In fact, any cut point divides the domain of one attribute in two partitions having approximately the same class distribution as the entire domain. Thus a condition on a single attribute does not improve class separation, so the two attributes will be discretized into two unique intervals, and the solution for this classification problem cannot be found. For this reason univariate supervised discretization methods based on information class entropy are unlikely to work. This suggests that in some cases the use of an univariate discretization method could negatively affect the accuracy of the learned theory.

An elegant and robust approach for overcoming this drawback is provided by evolutionary algorithms, which can be used for performing local multivariate discretization during the evolutionary learning process. Recent methods based on this approach deal with continuous attributes by means of inequalities. These methods differ among each other mainly in the way inequalities, describing continuous attribute subranges, are encoded, and in the definition of suitable genetic operators for modifying inequalities. A brief description of these methods is presented in section 6.6.

Evolutionary learners for ILP, like ECL REGAL, G–NET and DOGMA, generally treat continuous attributes as nominal ones or discretize them prior to induction, e.g., using Fayyad & Irani's algorithm. SIA01 adopts some method for handling numerical attributes, as it can randomly create and extend an interval for a numerical value.

6.2 ECL-LUD

The first method we propose for dealing with numerical values is named ECL-LUD (Divina et al., 2003c). With ECL-LUD (ECL with Local Unsupervised Discretization), we handle numerical attributes by using inequalities of the

form $a \leq X \leq b$, where X is a variable relative to a numerical attribute, and a, b are attribute values. An inequality for a numerical attribute is generated when that attribute is selected. Inequalities are then modified during the evolutionary process by using operators defined in the sequel. These operators use information on the distribution of the values of attributes in order to update the interval boundaries of inequalities. Information about the distribution of the values of numerical attributes is obtained by clustering the values of each attribute using a mixture of Gaussian distributions. To this end we use the Expectation-Maximization (EM) algorithm (Dempster et al., 1977).

For each numerical attribute the EM algorithm returns n clusters described by mean μ_i and standard deviation σ_i, $1 \leq i \leq n$ of Gaussian distributions. A begin b_{cl_i} and an end e_{cl_i} of a cluster cl_i are generated by intersecting the distributions of cl_i with the ones of cl_{i-1} and cl_{i+1}. Special cases are $b_{cl_i} = -\infty$ and $e_{cl_i} = +\infty$. The boundaries a, b of each inequality $a \leq X \leq b$ are contained in one cluster. In the following $P(a \leq X \leq b) = P(a, b)$ indicates the area under the curve of the Gaussian distribution between a and b.

6.2.1 Operators

Within each cluster, we use inequalities for restricting the range of values of an attribute variable.

An inequality can be modified by the following operations:

1. enlarge the boundaries;
2. shrink the boundaries;
3. shift the boundaries;
4. change cluster of the boundaries;
5. ground the inequality (i.e., restrict its range to a single value).

Formally, consider the inequality $I : a \leq X \leq b$, and let b_{cl} and e_{cl} be the begin and the end of the cluster cl containing I, respectively.

Enlarge This operator applied to I returns $I' = a' \leq X \leq b'$ where $a' \leq a$ and $b \leq b'$. The new bounds a', b' are computed in the following way:

1. let $min = minimum\,\{P(b_{cl} \leq X \leq a), P(b \leq X \leq e_{cl})\}$ the minimum of the probability that X is between b_{cl} and a and the probability that X is between b and e_{cl}.
2. generate randomly p with $0 \leq p \leq min$;
3. find two points a', b' such that $p = P(a' \leq X \leq a) = P(b \leq X \leq b')$.

Bounds are enlarged by generating probabilities instead of random points inside the cluster because in this way we can exploit the information about the distribution of the data values in an interval.

Shrink This operator applied to I returns $I' = a' \leq X \leq b'$ where $a' \geq a$ and $b' \leq b$. a' and b' are computed by randomly choosing $p \leq P(a, b)$ such that $p = P(a \leq X \leq a') = P(b' \leq X \leq b)$, and $a' \leq b'$.

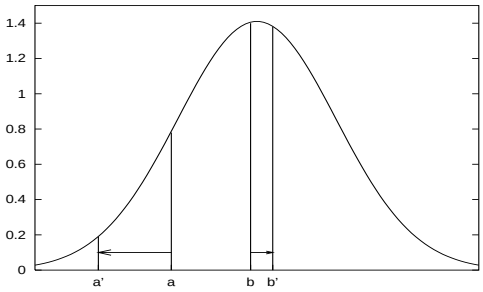


Figure 6.2: An example of the application of the Enlarge operator.

Example 6.2 Let $I = a \leq X \leq b$ the inequality describing the interval $[a, b]$ shown in figure 6.2. Then an example of the application of the enlarge operator is shown in figure 6.2. The cluster which I belongs to is represented by the normal distribution plotted in the figure. Two points a' and b' are generated, where $P(a' \leq X \leq a) = P(b \leq X \leq b')$. The resulting enlarged inequality is $I' = a' \leq X \leq b'$. An application of the shrink operator to I' could generate again I.

Shift This operator, applied to I returns $I' = a' \leq X \leq b'$ where a', b' are points in the cluster containing a, b such that $P(a' \leq X \leq b') = P(a \leq X \leq b)$.

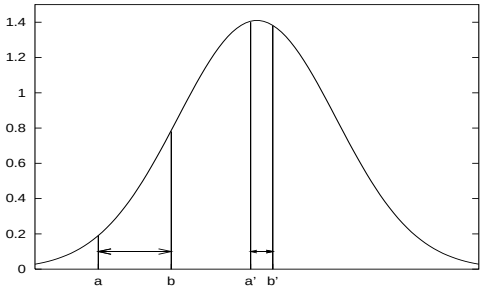


Figure 6.3: An example of the application of the Shift operator.

Example 6.3 Let $I = a \leq X \leq b$ the inequality describing the interval $[a, b]$ shown in figure 6.3. Then an application of the shift operator may

return the inequality $I' = a' \leq X \leq b'$, where $P(a \leq X \leq b) = P(a' \leq X \leq b')$.

Change Cluster This operator, applied to $I = a \leq X \leq b$ returns $I' = a' \leq X \leq b'$ where a', b' belong to a different cluster. First, a new cluster is chosen at random. Next a pair a', b' in the cluster with $a' \leq b'$ is randomly generated. (In general, $P(a' \leq X \leq b')$ is not equal to $P(a \leq X \leq b)$).

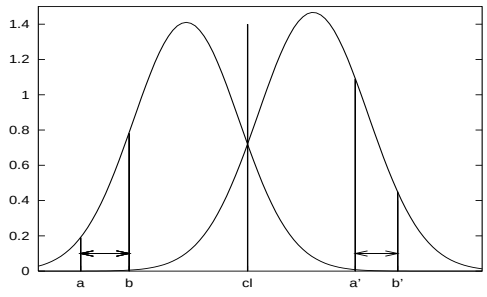


Figure 6.4: An example of the application of the Change cluster operator.

Example 6.4 Let $I = a \leq X \leq b$ the inequality describing the interval $[a, b]$ shown in figure 6.4. Let in the figure "cl" be the end and the begin of the clusters in which points a, b and a', b' are contained, respectively. Then an application of the change cluster operator applied to I can return the inequality $I' = a' \leq X \leq b'$.

Ground This operator, applied to I returns $I' = a' \leq X \leq a'$, with a' in the cluster containing a, b.

6.2.2 Incorporation of the Method into ECL

When a new clause is built using a positive example as a seed, or when a clause is specialized, atoms of the background knowledge are added to its body. Each time an atom containing a numerical argument is introduced in a clause, an inequality relative to that argument is added to the clause as well. For example, consider the following clause for example $c23$:

$$C = target(c23) : -q(c23, a), t(c23, y).$$

Suppose now that we would like to add the atom $r(c23, 8.23)$ stating that example $c23$ is in a relation r with a numerical value, in this case 8.23. Then we obtain the clause:

$$target(c23) : -q(c23, a), t(c23, y), r(c23, X), 8.23 \leq X \leq 8.23.$$

The operators for handling inequalities, introduced in the previous section, are used as mutation operators. When an operator is chosen then it is applied to an inequality $n\_choices$ times, where $n\_choices$ is a user supplied parameter. In this way $n\_choices$ new inequalities are generated and the one yielding the best fitness improvement is chosen.

Shrink and Ground These two operators are applied when specializing a clause. More precisely, when the system is specializing a clause by turning a variable into a constant, if the selected variable occurs in an inequality then either Shrink or Ground are applied to that inequality.

Enlarge This operator is applied when the system decides to generalize a clause. ECL has two generalization operators: *delete an atom* and *constant into variable* operators. When *delete an atom* is selected and the atom chosen for deletion describes the value of a numerical attribute, then both the atom and the inequality relative to the described attribute are deleted. If *delete an atom* is not selected and there are inequalities in the body of the clause chosen for mutation, then the system randomly selects between the *constant into variable* and *enlarge* operators.

Change Cluster and Shift These operators are applied with a probability pc (typical value 0.2), if in the selected individual there are inequalities.

The shrink and ground operators are specialization operators. In fact, if the shrink or the ground operator is applied to an inequality I contained in C, then for the resulting clause C', we have that $p'_C \subseteq p_C$ and $n_{C'} \subseteq n_C$. And from definition 2.9 we can conclude that C' is more specific than C.

On the other hand, the enlarge operator is a generalization operator. In fact if C is the clause to which the enlarge operator is applied, and C' is the resulting clause, then $p_C \subseteq p_{C'}$ and $n_C \subseteq n_{C'}$.

The other two operators, change cluster and shift, can be viewed as exploration operators. In fact it is impossible to predict the effects that an application of one of these two operators can have on the set of examples covered by a clause.

ECL-LUD is an unsupervised local discretization method. Information about the distribution of examples is obtained through a number of Gaussian distributions that describes an equal number of clusters for each numerical attribute. Other clustering algorithms, e.g., K-means clustering (MacQueen, 1967) or Minimal Spanning Tree clustering algorithm , could be used for obtaining clusters. However the EM algorithm has the desirable feature of describing the clusters by means of Gaussian distributions, which are used in the operators adopted by ECL-LUD for modifying inequalities.

6.3 Boundary Points

Boundary points represent candidate cut–points for discretization. They are used in the Fayyad & Irani's discretization algorithm, described in the following section, as well as in the methods developed for ECL and described in section 6.5. Boundary points have been introduced and analyzed in (Fayyad and Irani, 1992). A boundary point is defined as follows:

Definition 6.1 Given a numeric attribute A and a set of positive and negative examples, the values of A occurring in the examples are sorted in increasing order. A *boundary point* is the midpoint of two successive values of A occurring in examples of different classes.

Here we call boundary points also the smallest and biggest value of A, denoted by $-\infty$ and ∞, respectively.

Each pair of consecutive boundary points describes an interval, which can be of three types: *negative* if its values occur only in negative examples, *positive* if they occur only in positive examples, and *mixed* if the interval contains just one value, and this value occurs both in a positive and a negative example. An example is shown in figure 6.5.

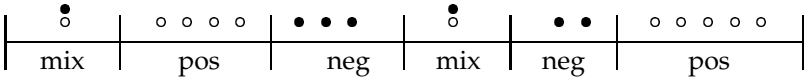


Figure 6.5: An example of boundary points of an attribute: ∘ denotes a value occurring in a positive example, while • a value occurring in a negative one.

Definition 6.2 Let A be a numerical attribute. We then define the set of boundary points relative to A, and we indicate it as $BP(A)$, as the sequence of boundary points of A sorted in increasing order.

Definition 6.3 Let A be a numerical attribute, and $BP(A)$ the relative set of boundary points. We then define a BP *interval* as an interval defined by two successive elements of $BP(A)$.

Boundary points are sufficient for finding the minimum of class information entropy, a measure used in the Fayyad & Irani's discretization algorithm.

6.4 Fayyad & Irani's Discretization

A standard approach for dealing with numerical attributes is to discretize them into intervals, and then use the intervals instead of the numerical values. To this end, the Fayyad & Irani's algorithm is widely used. In (Dougherty et al.,

1995; Kohavi and Sahami, 1996) a study of some discretization methods is conducted, and it is shown that Fayyad & Irani's discretization method represents a good way for globally discretizing numerical attributes.

This supervised recursive algorithm uses the class information entropy of candidate intervals to select the boundaries of the discretization intervals. Given a set S of instances, an attribute p, and a partition bound t, the class information entropy of the partition induced by t is given by:

$$E(p,t,S) = Entropy(S_1)\frac{|S_1|}{|S|} + Entropy(S_2)\frac{|S_2|}{|S|} \tag{6.1}$$

where S_1 is the set of instances whose values of p are in the first half of the partition and S_2 the set of instances whose values of p are in the second half of the partition. Moreover $|S|$ denotes the number of elements of S and $Entropy$ is defined as:

$$Entropy(S) = -p_+ \cdot log_2(p_+) - p_- \cdot log_2(p_-) \tag{6.2}$$

with p_+ and p_- the proportion of positive and negative examples in S respectively.

For a given attribute p the boundary t^* which minimizes $E(p,t,S)$ is selected as a binary discretization boundary. The method is then applied recursively to both the partitions induced by the selected boundary t^* until a stopping criterion is satisfied. The MDL principle is used to define the stopping criterion. Recursive partitioning within a set of instances stops if $Entropy(S) - E(p,t,S)$ is smaller than $log_2\frac{(N-1)}{N} + \frac{\Delta(p,t,S)}{N}$, where $\Delta(p,t,S) = log_2(3^k - 2) - [k \cdot Entropy(S) - k_1 \cdot Entropy(S_1) - k_2 \cdot Entropy(S_2)]$ and k_i is the number of class labels represented in S_i.

The algorithm works without a predefined number of intervals. Instead, it recursively splits intervals at the cut–point that minimizes the entropy, until the entropy decrease is smaller than the increase of MDL induced by the new point.

The method is applied to each numerical attribute of the domain. The attributes are split into a number of intervals, and each interval is considered as one value of a nominal attribute. If for treating numerical values, we use the intervals found with the Fayyad & Irani's algorithm, then we refer to ECL as ECL-GSD (ECL with Global univariate Supervised Discretization). So in ECL-GSD a number of intervals are used as nominal values for each numerical attribute.

Definition 6.4 Let A be a numerical attribute. Then we define the set of discretization points relative to A, and we indicate it as $DP(A)$, as the sequence of discretization points obtained using the Fayyad & Irani's algorithm sorted in increasing order.

Definition 6.5 Let A be a numerical attribute and $DP(A)$ the relative set of discretization points. We then define *DP interval* as an interval defined by two consecutive elements of $DP(A)$.

6.5 ECL-LSDc and ECL-LSDf

ECL-LUD is an unsupervised local discretization method. For this reason it can not take advantage of the class information of examples. In order to change an inequality, ECL-LUD takes into consideration only the density distribution as estimated by the Gaussian distribution of the relative cluster.

In order to exploit class information, we here introduce a local supervised method for dealing with numerical values in ECL. Also this method handles numerical attributes by means of inequalities of the form $l < A \leq u$, where $l < u$ are specific elements of $BP(A)$. However, inequalities are initialized and modified in a different way from how the same operations are performed in ECL-LUD. An element of $BP(A)$ is called *left-good* if it is not the left boundary of a negative $BP(A)$ interval, and *right-good* if it is not the right boundary of a negative $BP(A)$ interval. For example, in figure 6.6, t_0, t_1, t_3, t_5 are left-good, and t_1, t_2, t_4, t_6 are right good. We will consider only inequalities $l < A \leq u$ with l and u left- and right-good, describing intervals that do not start or end with a negative $BP(A)$ interval.

Example 6.5 Assume the boundary points of A are those in figure 6.6. Then $t_0 < A \leq t_2$ is a legal inequality, while $t_1 < A \leq t_3$ is not legal, because it describes an interval that ends with a negative $BP(A)$ interval.

Now assume $BP(A) = (t_0, \ldots, t_n)$, and consider an inequality $t_i < A \leq t_j$ with $i, j \in [0, n]$, $i < j$ and t_i, t_j left- and right-good. We introduce the following generalization and specialization operators.

enlarge

1. Randomly select either t_i or t_j.
2. (a) If t_i has been chosen, find the greatest $t_{i'}$ such that $i' < i$ and $t_{i'}$ is left-good. Set t_i to $t_{i'}$. If such $t_{i'}$ does not exist (if $i = 0$ or all intervals to the left of t_i are negative) then choose t_j and go to step *(b)* if it was not already tried.
   (b) If t_j has been chosen, find the smallest $t_{j'}$ such that $j' > j$ and $t_{j'}$ is right-good. Set t_j to $t_{j'}$. If such $t_{j'}$ does not exist (if $j = n$ or all intervals to the right of t_j are negative) then choose t_i and go to step *(a)* if it was not already tried.

shrink This operator is applicable if $|i - j| > 1$.

1. Randomly select either t_i or t_j.
2. (a) If t_i has been chosen, find the smallest $t_{i'}$ such that $i' < j, i' > i$ and $t_{i'}$ is left-good. Set t_i to $t_{i'}$.
   (b) If t_j has been chosen, find the greatest $t_{j'}$, where $j' < j, j' > i$ and $t_{j'}$ is right-good. Set t_j to $t_{j'}$.

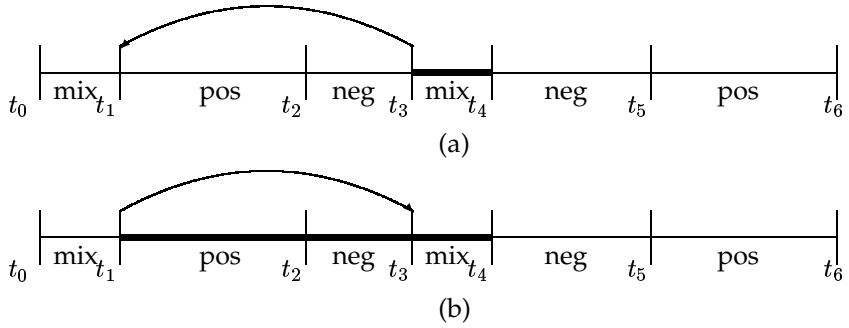


Figure 6.6: part (a): application of enlarge; part (b): application of shrink. The inequality is represented by a thick segment.

Notice that application of enlarge and shrink preserves the left- and right-goodness of the boundaries of an inequality.

Example 6.6 Figure 6.6 illustrates the application of the enlarge and shrink operators to inequalities represented by the thick lines. Enlarge applied to $t_3 < A \leq t_4$ shifts its left boundary t_3 to t_1, while shrink applied to $t_1 < A \leq t_4$ shifts its left boundary t_1 to t_3.

6.5.1 Incorporation of the Method into ECL

Two variants of the proposed method are introduced: ECL-LSDf (ECL with Local Supervised Discretization with Fine grain initialization) and ECL-LSDc (ECL with Local Supervised Discretization with Coarse grain initialization). ECL-LSDf and ECL-LSDc differ in the way inequalities are initialized in the two methods. For example, assume the clause

$$target(c) : -q(c, a), t(c, b).$$

is constructed with the example $target(c)$ as seed. If the specialization operator "add an atom" is chosen and the BK fact $r(c, 8.23)$ is selected, then the clause becomes

$$target(c) : -q(c, a), t(c, b), r(c, X), l < X \leq u.$$

where in ECL-LSDf l, u are boundaries of the BP interval containing 8.23, while in ECL-LSDc l, u are boundaries of the DP interval containing 8.23.

The same operators for evolving rules are used in both ECL-LSDf and ECL-LSDc. If the generalization operator "delete an atom" is chosen and $r(c, X)$ is selected for deletion, then $r(c, X)$ and the corresponding inequality are removed from the clause. The other possible generalization operator consists of a random choice between the "constant into variable", which replaces one of the constants a, b, c with a variable, and the "enlarge" operator, which enlarges one boundary of the inequality. If the specialization operator "variable into constant" is chosen and the variable X is selected, then the "shrink" operator is applied to the relative inequality.

6.6 Related Work

Recent methods based on evolutionary algorithms performing multivariate discretization during the learning process are (Kwedlo and Kretowski, 1999; Bacardit and Garrel, 2002; Bacardit and Garrel, 2003), where the EAs for classification GIL (Janikow, 1993) and GABIL (De Jong et al., 1993) are extended into the systems EDRL-MD and GAssit, respectively. Both EDRL-MD and GAssist are Pittsburgh EAs and deal with continuous attributes by means of inequalities that can be modified during the evolutionary process.

In EDRL-MD, candidate solutions are encoded by means of string chromosomes. The string is composed by n substrings, each encoding a condition related to one attribute. In case of continuous attributes the relative substring encodes the lower and the upper thresholds of an interval describing the allowed subrange of values for the described attribute.

GAssist evolves individuals that are ordered variable–length rule sets. The knowledge representation for real–valued attributes is called Adaptive Discretization Intervals rule representation (*ADI*). This representation uses the same semantics for rules as GABIL (Conjunctive Normal Form predicates), but uses non–static intervals formed by joining several neighbor discretization intervals. The representation can also combine several discretizations at the same time, allowing the system to choose the correct discretizer for each attribute.

Another EA system adopting a similar method for dealing with numerical value is HIDER* (Giráldez et al., 2003; Aguilar-Ruiz et al., 2002; Giráldez et al., 2004). HIDER* utilizes the USD (Giráldez et al., 2002) discretizer in order to find a number of boundary points that are used as limits of intervals describing subranges of values for numerical attributes. The USD discretizer divides the domains of 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, USD 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. The main features of the USD are: it is deterministic, does not need any user–parameter and its complexity is sub-quadratic. An important feature of HIDER* is its encoding method: each attribute is encoded with only one gene, reducing considerably the length of the individuals, and therefore the search space size, making the algorithm faster while maintaining its prediction accuracy. In this encoding inequalities are represented as natural numbers, and can be easily modified during the evolutionary process.

To our knowledge, the only EA for ILP that adopts some methods for dealing with numerical values is SIA01 (Augier et al., 1995) (see chapter 4). SIA01 uses intervals for numerical attributes, which are randomly created and modified during the evolutionary process.

Discretization is not the only way to handle real–valued attributes in EC applied to ICL. Some examples of alternative ways are induction of decision trees (either axis–parallel or oblique), by either generating a full tree by means

of genetic programming operators, as it happens in GALE (Llorá and Garrell, 2001b; Llorá and Garrell, 2001a) or using a heuristic method to generate the tree and later a genetic algorithm or an evolutionary strategy to optimize the test performed at each node (Cantu-Paz and Kamath, 2003). Another example is represented by the XCS system (Stone and Bull, 2003; Wilson, 1998). XCS induces rules with real–valued intervals represented as a (c_i, r_i), where c_i and r_i are real numbers, which represents the center and radius of the interval $[c_i - r_i, c_i + r_i]$. Another strategy is generating an instance set used as the core of a *k-NN* classifier (Llorá and Garrell, 2001b).

In this chapter we have seen some examples of discretization algorithms. In the literature, several other discretization algorithms are reported. Among these the Mántaras discretizer (De Mántaras, 1991) which is similar to the Fayyad & Irani's algorithm, but uses a different formulation of the entropy minimization. Another example of discretization method similar to the Fayyad & Irani's, but not relying on entropy, is represented by the Holte's discretization method (Holte, 1993). This methods, used in IB1 (Aha et al., 1991), attempts to divide the domain of every continuous attribute into pure bins, each containing a strong majority of one particular class with the constraint that each bin must include at least some pre specified number of instances. Yet another example is ChiMerge (Kerber, 1992). This discretizer creates an initial pool of cut points containing the real values in the domain to discretize, and iteratively merges neighbor intervals that make true a certain criterion based on the χ^2 statistical test.

6.7 Conclusions

In ILP discretization methods based on entropy, e.g., Fayyad & Irani's algorithm, can be negatively affected by the presence of examples having multiple values for a numerical argument, as we have seen in example 6.1. In order to overcome this phenomenon we introduced three methods for dealing with numerical values in ECL. When in ECL intervals found with the Fayyad & Irani's algorithm are used for dealing with numerical values, we refer to ECL with ECL-GSD. In this case the standard mutation operators of ECL, described in chapter 5, are used, because numerical values are treated as nominal ones.

An overview of the discretization methods presented in this chapter is given in table 6.1. ECL-LUD, ECL-LSDf and ECL-LSDc treat numerical values by means of inequalities. The way in which inequalities are initialized and modified during the evolutionary process is different.

The first method, called ECL-LUD, is a local unsupervised discretization method. It first finds a number of clusters for each numerical attribute, by means of the EM algorithm. Each cluster is described by a Gaussian distribution, which represents the density distribution of examples in the cluster. Inequalities are initialized to a single value, and can be then modified by means of ad-hoc mutation operators. Some of these operators exploit information regarding the density distribution of examples. ECL-LSDf and ECL-LSDc are al-

Method	Class info	G/L	Initialization	Operators
ECL-GSD	supervised	global	DP interval	standard
ECL-LUD	unsupervised	local	Inequality to single value	enlarge, shrink change cluster shift
ECL-LSDf	supervised	local	Inequality to BP interval	enlarge shrink
ECL-LSDc	supervised	local	Inequality to DP interval	enlarge shrink

Table 6.1: Overview of the components of the discretization methods described in this chapter. For each method, we indicate if the method exploits information about the class of examples, if the method is applied previously or during the evolutionary process, the initial range assigned to numerical variables, and the operators used for modifying this initial range.

ternative local supervised discretization methods. Both methods exploit class information of training examples. This is an advantage with respect to ECL-LUD which exploits only information about the density distribution of examples. ECL-LSDf and ECL-LSDc use the same operators for modifying inequalities during the evolutionary process. The difference between the two methods lies in the way inequalities are initialized. In fact ECL-LSDc initializes inequalities to already sub–optimal intervals, which are found with the application of the Fayyad & Irani's algorithm, while ECL-LSDf initializes inequalities to a BP interval, which is not already sub-optimal. For this reason, ECL-LSDc can reach a good solution in a less amount of time than ECL-LSDf. ECL-LSDf needs more computational resources for reaching the same intervals as those described by inequalities of clauses obtained by ECL-LSDc. Moreover, ECL-LSDf runs the risk of overfitting the training examples. This is due to the fact that solutions found could contain many clauses, each of which contains inequalities describing intervals that are good relatively to the training examples, but that are rather poor relatively to testing examples.

We believe that the treatment of numerical values in EAs for ILP has not been given enough attention. To our knowledge, only SIA01 addressed this issue. In fact, SIA01 can create intervals for numerical values. However these intervals are randomly modified during the evolutionary process.

In the next chapter we present an experimental comparison of the four methods.

Chapter 7

Experimental Evaluation

In this chapter we present an experimental evaluation of the various components of ECL introduced in chapters 5 and 6.

The first aspect we want to investigate, is how the optimization phase and greediness in the mutation operators affects both the quality of the solution found and the computational time required by the evolutionary process.

A second aspect concerns the use of different amounts of background knowledge at each iteration performed by ECL. The possibility of using only a subset of the background knowledge at each iteration was introduced for reducing the computational time required by the evaluation of individuals. We want to assess how this affects the quality of the solution found and the computational time required by the learning process.

A third aspect regards the effectiveness of the variants of the US selection mechanism introduced in chapter 5. In particular we are interested in assessing the effectiveness of the various selection mechanisms in promoting diversity in the population as well as a good coverage of the positive examples.

In section 5.8, we stated that the procedure based on precision is more effective than the procedure based on the algorithm for solving weighted set covering problems for extracting a solution from the final population. We want to experimentally confirm that statement.

The last set of experiments for evaluating the components of ECL, are aimed at assessing the effectiveness of the various methods for treating numerical values presented in chapter 6. In all the experiments we use both propositional and relational datasets.

This chapter is structured in the following way. In section 7.1 we begin by giving the setting used in the experiments. Section 7.2 reports a first set of experiments aimed at verifying the effectiveness of incorporating the optimization phase and a degree of greediness in the mutation operators. In section 7.3 we present results obtained by ECL using different values of the parameter pbk, that controls the amount of background knowledge used at each iteration. Results of experiments obtained with the different selection operators are presented in section 7.4. In section 7.5 experiments showing the different results

of ECL when different procedures for extracting a final solution are applied to the same final population are presented. Section 7.6 reports results on experiments using the different methods for handling numerical values. Finally we compare the results obtained by ECL with those obtained by other systems for inductive concept learning.

7.1 Experimental Settings

Dataset	Examples (+,-)	Continuous	Nominal	BK size
Australian	690 (307,383)	6	8	9660
Breast	699 (458,241)	10	0	6275
Crx	690 (307,383)	6	9	10283
Echocardiogram	74 (24,50)	5	1	750
German	1000 (700,300)	24	0	24000
Glass2	163 (87,76)	9	0	1467
Heart	270 (120,150)	7	6	3510
Hepatitis	155 (123,32)	6	13	2778
Liver	345 (145,200)	6	0	2070
Ionosphere	351 (225,126)	34	0	11934
Pima-Indians	768 (500,268)	8	0	6144
Sonar	208 (97,111)	60	0	12480
Vote	435 (267,168)	0	16	6568
Wdbc	569 (212,357)	30	0	17070
Wpbc	198 (47,151)	33	0	6530

Table 7.1: Characteristics of the propositional datasets used in the experiments performed in this chapter. From left to right: number of examples (positive, negative), of continuous attributes, of nominal attributes, and of ground facts in the BK.

In all the experiments, unless stated otherwise, we use ten–fold cross validation. Each dataset is divided in ten disjoint sets of similar size. One of these sets forms the test set, and the union of the remaining nine the training set. The system is run three times, using different random seeds, on each training set and its output (a Prolog program) is evaluated on the corresponding test set (so the algorithm is run 30 times per dataset).

Table 7.1 presents the features of the propositional datasets used in this chapter. These datasets are taken from the UCI Machine Learning repository (Blake and Merz, 1998), and are well known benchmarks. They regard various problems, going from detecting frauds in the use of credit cards, to detecting if a person is likely to develop diabetes, and so on.

Table 7.2 shows the features of the relational datasets. The mutagenesis dataset (Debnath et al., 1991) originates from the problem in organic chemistry of

learning the mutagenic activity of nitroaromatic compounds. The traffic dataset (Džeroski et al., 1998a; Džeroski et al., 1998b) concerns the task of detecting sections of roads where a traffic problem - an *accident* or a *congestion* - has occurred at a specific time. The biodegradability dataset (Džeroski et al., 1999) originates from the task of predicting the half-life time in water for aerobic aqueous biodegradation of a compound. It consists of four classes: *fast* if the biodegradation time of a compound is up to 7 days, *moderate* if the biodegradation time is 1 to 4 weeks, *slow* if the biodegradation time it 1 to 6 months, and *resistant* in the other cases. On the biodegradability dataset we used the same splitting of data as in (Džeroski et al., 1999), consisting of five different ten–fold cross validation sets. The pyrimidines dataset originated from (King et al., 1995) and regards the classic drug design problem of inhibition of Dihydrolate Reductase by pyrimidines. Pyrimidine compounds are antibiotics. For this dataset we used a five–fold cross validation used in the original study of the database.

Dataset	Examples	Continuous	Nominal	BK size
Mutagenesis	188 (125,63)	6	4	13125
Traffic	256 (62,66,128)	3	2	15770
Biodegradability	328 (65,120,101,42)	2	4	17260
Pyrimidines	2788 (1394,1394)	0	8	2116

Table 7.2: Characteristics of the relational datasets. From left to right: dataset name, total number of examples and, between brackets, number of examples per class, number of continuous and nominal properties and number of facts in the BK.

The parameters used in the experiments for both the propositional and the relational datasets are shown in table 7.3. All these parameters were obtained after few, in the order of 10, runs of the system on the relative dataset. We emphasize that the parameter settings chosen were the ones which led to the best classification accuracy in the training set, i.e., the test set was never accessed during the runs allocated for parameter setting. If not differently stated, ECL-LSDc is used. The reasons for this choice will become evident in section 7.6. Naturally, in the experiments regarding the methods for dealing with numerical values, all the methods are tested.

7.2 Experiments on Incorporating Greediness in ECL

In this section we want to evaluate the effectiveness of incorporating the optimization phase that follows the mutation and of using the non–random mutation operators described in chapter 5. In order to do so, we perform experiments with ECL in three settings:

ECL-GA In this setting ECL is run with all the values of N_i set to 1 and with no

Dataset	pop size	gen	sel	max_iter	Ni	lc	pbk
Australian	50	10	15	1	(4,4,4,4)	6	0.4
Biodegradability	50	10	10	1	(4,4,4,4)	4	1.0
Breast	50	5	5	1	(3,3,3,3)	5	1.0
Crx	80	20	15	1	(4,4,4,4)	7	1.0
Echocardiogram	40	8	10	10	(4,4,4,4)	4	0.7
German	200	10	10	2	(3,4,3,4)	6	0.4
Glass2	150	15	20	3	(2,8,2,9)	5	0.8
Heart	50	10	15	1	(4,4,4,4)	6	1.0
Hepatitis	50	10	10	5	(4,4,4,4)	7	0.2
Ionosphere	50	10	15	6	(4,8,4,8)	6	0.2
Liver	60	10	7	1	(2,5,3,5)	4	0.2
Mutagenesis	50	10	15	2	(4,8,2,8)	3	0.8
Pima-Indians	60	10	7	5	(2,5,3,5)	4	0.2
Pyrimidines	70	25	15	1	(4,2,5,2)	4	1.0
Sonar	80	10	15	1	(4,8,4,8)	5	1.0
Traffic	30	10	10	1	(10,2,2,2)	8	1.0
Vote	80	10	15	2	(3,6,2,5)	4	0.5
Wdbc	100	15	15	1	(3,7,5,3)	2	1.0
Wpbc	20	5	5	1	(3,3,3,3)	5	1.0

Table 7.3: Parameter settings used in the experiments: *pop size* is the population size, *gen* is the number of generations performed by the GA, *sel* is the number of individuals selected per generation, *max_iter* is the maximum number of iterations performed, N_i, $i \in [1, 4]$, are the greediness parameters of the mutation operators, *lc* is the maximum length of a clause, and *pbk* is the probability of selecting a BK fact.

optimization phase. In this way all the mutation operators act randomly, as in standard GA operators;

ECL-NOT In this setting ECL is run with the values of N_i set as reported in table 7.3. The optimization phase is not performed.

ECL-Opt In this setting ECL is run with the parameters N_i set as reported in table 7.3. The optimization phase is performed after mutation, with a maximum of 10 optimization steps.

In order to perform a fair comparison, we increased the values of the parameter sel in ECL-GA and in ECL-NOT so that the three settings perform about the same number of evaluations.

Table 7.4 reports the average results obtained by ECL in the three different settings.

A first aim of these experiments was to assess how the incorporation of the optimization phase and the use of greedy mutation operators affect the computational time required by the evolutionary process. From the experiments,

Dataset	Setting	Accuracy	Time (s)	Simplicity
Accidents	ECL-GA	0.82 (0.01)	**2752.16 (82.69)**	35.3 (10.92)
	ECL-NOT	0.88 (0.01)	3092.62 (103.72)	23.37 (9.69)
	ECL-Opt	**0.95 (0.02)**	3395.01 (136.82)	**3.55 (0.49)**
Australian	ECL-GA	0.82 (0.03)	**1353.02 (7.72)**	14.4 (2.77)
	ECL-NOT	0.83 (0.03)	1444.05 (16.54)	12.30 (2.58)
	ECL-Opt	**0.85 (0.01)**	1686.38 (144.07)	**6.10 (2.18)**
Breast	ECL-GA	0.92 (0.01)	**173.67 (2.33)**	11.30 (2.20)
	ECL-NOT	0.93 (0.02)	238.72 (11.52)	11.50 (2.01)
	ECL-Opt	**0.95 (0.02)**	286.13 (37.00)	**8.60 (0.41)**
Congestions	ECL-GA	0.91 (0.02)	**2532.98 (98.43)**	5.70 (1.25)
	ECL-NOT	0.92 (0.02)	2983.15 (38.65)	5.46 (1.46)
	ECL-Opt	**0.94 (0.02)**	3246.30 (138.73)	**3.95 (0.35)**
Crx	ECL-GA	**0.84 (0.04)**	**1707.82 (66.47)**	13.30 (3.34)
	ECL-NOT	0.83 (0.03)	1852.97 (54.47)	11.70 (3.13)
	ECL-Opt	**0.84 (0.01)**	2668.00 (176.45)	**4.80 (0.05)**
Echocardiogram	ECL-GA	0.70 (0.03)	**1245.21 (6.21)**	**2.50 (0.53)**
	ECL-NOT	0.73 (0.03)	1311.95 (10.31)	**2.50 (0.53)**
	ECL-Opt	**0.74 (0.01)**	1443.63 (36.62)	2.60 (0.70)
German	ECL-GA	**0.74 (0.02)**	**1041.74 (19.83)**	14.2 (2.20)
	ECL-NOT	0.73 (0.03)	1153.52 (11.32)	16.70 (3.09)
	ECL-Opt	**0.74 (0.01)**	1605.75 (144.34)	**11.70 (0.24)**
Glass2	ECL-GA	0.82 (0.04)	**956.17 (2.74)**	**3.90 (0.99)**
	ECL-NOT	**0.85 (0.03)**	1143.05 (21.49)	4.40 (1.51)
	ECL-Opt	**0.85 (0.01)**	1246.00 (55.94)	4.20 (1.23)
Heart	ECL-GA	0.77 (0.03)	**345.34 (8.40)**	9.20 (1.93)
	ECL-NOT	0.78 (0.02)	474.51 (13.41)	7.40 (2.12)
	ECL-Opt	**0.80 (0.03)**	436.38 (57.59)	**4.20 (1.32)**
Hepatitis	ECL-GA	0.82 (0.02)	**878.202 (6.31)**	13.00 (1.63)
	ECL-NOT	**0.83 (0.03)**	954.61 (10.34)	13.40 (1.51)
	ECL-Opt	**0.83 (0.02)**	1056.73 (63.84)	**7.60 (0.95)**
Ionosphere	ECL-GA	0.87 (0.04)	**4364.59 (15.96)**	25.90 (3.96)
	ECL-NOT	**0.89 (0.03)**	4498.72 (13.21)	25.10 (2.72)
	ECL-Opt	**0.89 (0.02)**	5276.83 (138.93)	**12.50 (1.48)**
Mutagenesis	ECL-GA	0.84 (0.03)	**407.88 (4.56)**	**4.56 (0.73)**
	ECL-NOT	0.86 (0.02)	470.32 (5.14)	4.70 (0.95)
	ECL-Opt	**0.88 (0.01)**	542.88 (27.88)	7.92 (1.51)
Pima-Indians	ECL-GA	0.73 (0.03)	**1031.71 (10.95)**	9.70 (3.06)
	ECL-NOT	0.74 (0.02)	1157.64 (14.47)	9.20 (1.87)
	ECL-Opt	**0.76 (0.01)**	1214.75 (31.86)	**8.40 (1.84)**
Vote	ECL-GA	0.92 (0.03)	**768.78 (42.61)**	10.20 (2.55)
	ECL-NOT	0.93 (0.01)	967.38 (56.68)	9.00 (2.91)
	ECL-Opt	**0.94 (0.02)**	993.24 (43.29)	**3.70 (1.06)**

Table 7.4: Experiments with various setting of greediness of ECL. In ECL-GA ECL runs as a standard GA. In ECL-NOT greedy mutation operators are used and in ECL-Opt both greedy mutation operators and the optimization phase are used.

it emerges that the computational time required by ECL-GA and by ECL-NOT is smaller than the time required by ECL-Opt. In particular ECL-GA is the fastest setting. This result was expected, since in ECL-GA mutations are done randomly and no optimization phase takes place.

The second, and main, aim of these experiments was to establish if the incorporation of the optimization phase and of the greedy mutation operators was beneficial for improving the accuracy of the found solutions.

It can be noticed that ECL-Opt generally obtained the best accuracies. Only on the Crx dataset ECL-GA obtained the same accuracy as the one obtained by ECL-Opt, but with a higher standard deviation. It can also be noticed that generally ECL-NOT obtained better results than ECL-GA. In some cases the solution found by ECL-NOT is equal to the solution found by ECL-Opt. However neither ECL-GA nor ECL-NOT were capable of finding solutions with higher accuracy than the solutions found by ECL-Opt. This is evident especially for the relational datasets.

For the Traffic dataset, this is not only true for the single classes. Also when the induced clauses are used for addressing the whole problem, the incorporation of knowledge in ECL is beneficial for obtaining better results. In fact, on the Traffic dataset, ECL-GA obtained an accuracy of 0.74, ECL-NOT of 0.83 while ECL-Opt obtained the best accuracy of 0.93. This is due to the fact that for the relational datasets it is important to find good relations among the arguments of a clause. It is more likely to find good relations with greedy operators and an optimization phase than using only random operators.

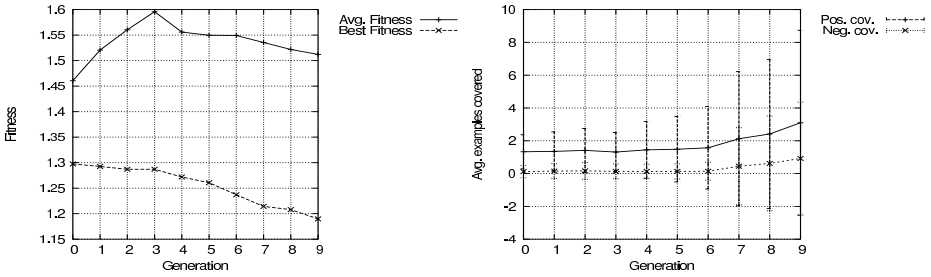
It is interesting to notice that generally the logic programs induced by the three settings become simpler with the use of greedy mutation operators and with the inclusion of the optimization phase. This is mostly due to the optimization phase. In fact during this phase individuals are rapidly refined so that less clauses are needed for obtaining a logic program with good accuracy. It can be seen that only on three datasets, namely on the Echocardiogram, the Glass2 and the Mutagenesis dataset, ECL-Opt did not find the simplest solution. Only on the Mutagenesis dataset the difference in the simplicity of the solution is significant, while in the other two cases the simplicity of the solutions is comparable. On all the other datasets ECL-Opt found the simplest solution. This can be noticed especially for the Accidents dataset where the solution found by ECL-Opt is almost 10 times simpler than the one obtained by ECL-GA and almost 7 times simpler than the one obtained by ECL-NOT.

In order to summarize the performance of the three settings of ECL and the significance of the results with respect to the accuracy, we compute the statistical paired two-tailed t-test, with confidence level of 1% and 5%. The t-test is performed on the 30 results obtained from the 10 folds and the 3 random seeds. For the Echocardiogram, the Glass2, the Heart and the Hepatitis datasets, the results are not normally distributed and so the t-test cannot be performed on these datasets. Table 7.5 reports the results of the t-test for the other datasets used in this section. Using a confidence level of 1%, we can see that ECL-Opt outperformed once ECL-GA, namely it obtained significantly better results on the Accidents dataset. If we extend the confidence level to 5%, we have that

	ECL-GA	ECL-NOT	ECL-Opt	Total
ECL-GA	–	0	0	0
ECL-NOT	0	–	0	0
ECL-Opt	1 (3)	0 (1)	–	1 (4)
Total	1 (3)	0 (1)	0	

Table 7.5: Results of the two-tailed paired t-test for the used datasets with 1% confidence level: each entry contains the number of datasets on which the algorithm in the row is significantly better than the one in the column. The results of the test using 5% confidence level are reported between brackets when they differ from those using 1% confidence level.

ECL-Opt outperforms ECL-GA on three datasets, namely the Accidents, the Congestion and the Mutagenesis datasets. Using a 5% confidence level we have that ECL-Opt outperforms ECL-NOT on the Accidents dataset.



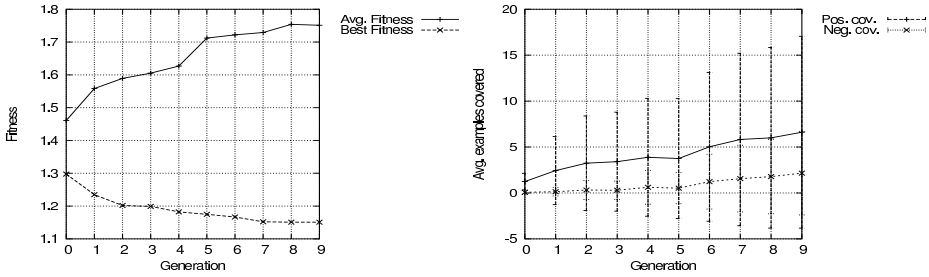
(a) Average and best fitness.

(b) Average coverage of individuals.

Figure 7.1: Graphs relative to fitness and coverage for 10 runs of ECL-GA on the Accidents dataset. Fitness is minimized.

The difference of performance between ECL-Opt and the other two settings is evident on the Accidents dataset. For this reason we want to analyze the dynamics of the three settings on this dataset. In graphs 7.1(a), 7.2(a) and 7.3(a), we show the best and average fitness of the population at every generation, computed over 10 runs of the various settings. In graphs 7.1(b), 7.2(b) and 7.3(b) we report the average number of positive and negative examples covered by an individual in the population evolved with the three settings, computed over 10 runs.

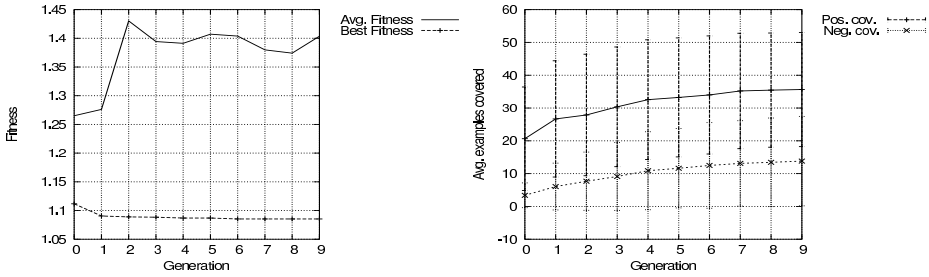
From the graphs, it can be seen that the fitness of the best individual in the population is better if more knowledge is incorporated in ECL. It is interesting to notice that the same does not hold for what concerns the average fitness. In fact the average fitness is better in the population evolved with ECL-GA than in the population evolved with ECL-NOT, and is comparable with the aver-



(a) Average and best fitness.

(b) Average coverage of individuals.

Figure 7.2: Graphs relative to fitness and coverage for 10 runs of ECL-NOT on the Accidents dataset. Fitness is minimized.



(a) Average and best fitness.

(b) Average coverage of individuals.

Figure 7.3: Graphs relative to fitness and coverage for 10 runs of ECL-Opt on the Accidents dataset. Fitness is minimized.

age fitness of the population evolved with ECL-Opt. By looking at the graphs relative to the coverage of each individual evolved in the three settings, it can be noticed that individuals evolved by ECL-GA are more specific than those evolved by ECL-Opt and by ECL-NOT. In particular individuals evolved by ECL-Opt are much more general than those evolved by the other two settings. This fact explains the difference in the simplicity of the solution found, which is much higher in ECL-GA and ECL-NOT than in ECL-Opt. So, on the Accidents dataset, incorporating more knowledge has the effect of allowing ECL-Opt to evolve more general individuals, that have on average not better performances on the training sets, but when used on the test sets yield better results.

This suggests that incorporating knowledge in the mutation operators and the optimization phase is beneficial in order to obtain more accurate solutions, especially for relational datasets.

7.3 Experiments on Background Knowledge Selection

In this section we present and discuss results obtained with different values of the parameter pbk. The aim of these experiments is to evaluate the behavior of ECL when portions of different sizes of the background knowledge are used at each iteration. With these experiments we want to evaluate the impact of the parameter pbk on both the computational time and the accuracy of the obtained solutions. To this end, for each dataset we run ECL using five values of pbk, namely: 0.2,0.4,0.6,0.8 and 1. When pbk is set to 1 the whole background knowledge is used. Recall that BK is randomly selected, where pbk controls the probability that each fact of the background knowledge has of being selected and used inside each iteration of ECL. Tables 7.6 and 7.7 report the results of the experiments.

From these results, it can be noticed that using a smaller portion of the background knowledge does not necessarily lead to worse results. On the opposite, in some cases the use of the whole background knowledge does not lead to better performance. As far as the accuracy of the solution is concerned, we cannot fix an optimal value of pbk, since this value is domain dependent. We can, however, observe that the use of more background knowledge does not generally yield to overfitting. Another observation that can be made is that for the relational datasets best performance are obtained with high values of pbk. This can be explained by the fact that for these datasets using too little background knowledge prevents ECL from finding good relations among the objects of the domain, thus leading to poor solutions.

Concerning the computational time, the results confirm that the computational cost of ECL increases proportionally to the values of pbk, as can be seen from figure 7.4. In the figure, the average time employed by ECL on all the benchmark datasets used in the experiments with different values of pbk is shown. The more background knowledge is used the less efficient ECL is.

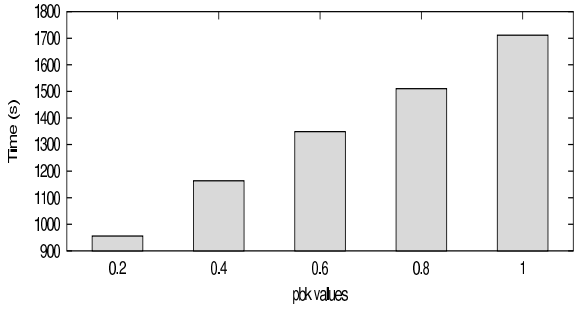


Figure 7.4: Average times, in seconds, employed by ECL with different values of pbk.

Dataset	pbk	Accuracy	Accuracy on train	Time (s)	Simplicity
Accidents	0.2	0.88 (0.03)	0.89 (0.01)	**492.0 (12.03)**	**2.4 (1.26)**
	0.4	0.87 (0.03)	0.90 (0.01)	564.15 (23.18)	3.9 (1.52)
	0.6	0.89 (0.02)	0.91 (0.02)	669.5 (31.73)	3.8 (1.03)
	0.8	0.93 (0.02)	0.95 (0.02)	966.8 (62.81)	5.2 (1.62)
	1.0	**0.95 (0.02)**	0.95 (0.02)	1083.8 (86.82)	3.55 (0.49)
Australian	0.2	0.85 (0.03)	0.84 (0.01)	**650.55 (29.36)**	**2.1 (0.99)**
	0.4	**0.85 (0.01)**	0.85 (0.02)	1266.83 (73.28)	5.3 (1.06)
	0.6	0.83 (0.04)	0.85 (0.02)	1531.8 (125.66)	6.1 (1.73)
	0.8	**0.85 (0.01)**	0.85 (0.02)	1686.38 (144.07)	6.1 (2.18)
	1.0	0.83 (0.01)	0.84 (0.03)	1729.57 (146.23)	6.2 (2.49)
Breast	0.2	0.93 (0.03)	0.94 (0.01)	**89.64 (4.32)**	**3.2 (0.79)**
	0.4	0.94 (0.02)	0.96 (0.01)	115.23 (6.31)	6.9 (2.33)
	0.6	**0.96 (0.02)**	0.96 (0.00)	144.17 (11.97)	9.1 (1.59)
	0.8	0.95 (0.02)	0.97 (0.01)	177.83 (14.73)	10 (2.94)
	1.0	0.95 (0.02)	0.96 (0.00)	286.13 (37.00)	8.60 (0.41)
Congestions	0.2	0.79 (0.03)	0.84 (0.03)	**525.80 (26.36)**	**1.7 (0.48)**
	0.4	0.83 (0.03)	0.86 (0.03)	597.43 (34.73)	2.5 (1.08)
	0.6	0.87 (0.02)	0.88 (0.04)	737.68 (71.03)	2.8 (0.79)
	0.8	0.90 (0.02)	0.93 (0.03)	981.49 (102.65)	3.5 (0.85)
	1.0	**0.94 (0.02)**	0.96 (0.01)	1107.30 (138.73)	3.95 (0.35)
Crx	0.2	0.83 (0.03)	0.84 (0.02)	**1446.41 (48.61)**	**2.6 (0.70)**
	0.4	**0.85 (0.03)**	0.85 (0.01)	2260.93 (152,31)	4.1 (1.10)
	0.6	0.84 (0.03)	0.86 (0.01)	2323.95 (144.27)	5.3 (1.83)
	0.8	**0.85 (0.03)**	0.86 (0.01)	2384.47 (161.58)	4.4 (1.65)
	1.0	0.84 (0.01)	0.86 (0.01)	2668.00 (176.45)	4.80 (0.05)
Echocardiogram	0.2	0.72 (0.03)	0.77 (0.02)	**1033.3 (29.10)**	**1.8 (0.63)**
	0.4	0.73 (0.03)	0.79 (0.02)	1206.84 (46.87)	2.2 (0.79)
	0.6	0.69 (0.04)	0.82 (0.02)	1384.3 (44.71)	**1.8 (0.79)**
	0.8	0.70 (0.03)	0.83 (0.01)	1545.5 (39.21)	3.2 (1.03)
	1.0	**0.78 (0.02)**	0.85 (0.02)	1666.42 (33.40)	3.7 (0.67)
German	0.2	0.72 (0.03)	0.75 (0.01)	**1232.61 (50.41)**	7.3 (1.89)
	0.4	**0.74 (0.01)**	0.76 (0.01)	1605.75 (144.34)	11.70 (0.24)
	0.6	**0.74 (0.02)**	0.75 (0.01)	2015.74 (53.8)	8.9 (2.56)
	0.8	**0.74 (0.02)**	0.75 (0.01)	2223.56 (70.59)	**6.0 (1.76)**
	1.0	**0.74 (0.01)**	0.74 (0.01)	2874.49 (138.37)	6.4 (1.43)
Glass2	0.2	0.81 (0.04)	0.81 (0.03)	**935.26 (41.32)**	**2.5 (0.71)**
	0.4	0.79 (0.05)	0.83 (0.02)	1025.79 (53.56)	2.7 (0.82)
	0.6	0.79 (0.04)	0.86 (0.01)	1047.95 (34.24)	2.7 (0.48)
	0.8	**0.85 ((0.01)**	0.86 (0.01)	1246.00 (55.94)	4.2 (1.23)
	1.0	0.80 (0.02)	0.89 (0.01)	1445.10 (39.31)	5.2 (1.03)
Heart	0.2	0.75 (0.05)	0.78 (0.02)	**326.89 (12.66)**	**2.3 (0.48)**
	0.4	**0.81 (0.02)**	0.81 (0.02)	360.47 (29.15)	4.7 (1.42)
	0.6	**0.81 (0.03)**	0.82 (0.02)	391.74 (36.88)	4.6 (0.97)
	0.8	0.79 (0.04)	0.83 (0.02)	429.20 (39.40)	5.5 (1.18)
	1.0	0.80 (0.03)	0.83 (0.03)	436.38 (57.59)	4.20 (1.32)

Table 7.6: First set of experiments with different values of pbk. pbk is the probability that a fact of the background knowledge is selected at each iteration of ECL.

Dataset	pbk	Accuracy	Accuracy on train	Time (s)	Simplicity
Hepatitis	0.2	**0.83 (0.02)**	0.88 (0.01)	**1056.73 (63.84)**	**7.60 (0.95)**
	0.4	0.82 (0.02)	0.94 (0.01)	1108.14 (57.05)	18.8 (2.30)
	0.6	0.81 (0.04)	0.95 (0.01)	1254.51 (18.58)	19.1 (3.48)
	0.8	0.80 (0.03)	0.94 (0.01)	1265.01 (20.62)	18.0 (2.05)
	1.0	**0.83 (0.03)**	0.94 (0.01)	1279.30 (13.41)	12.2 (1.55)
Ionosphere	0.2	**0.89 (0.02)**	0.93 (0.02)	**3200.99 (109.48)**	12.50 (1.48)
	0.4	0.86 (0.02)	0.91 (0.01)	3496.21 (64.80)	6.0 (1.05)
	0.6	0.88 (0.02)	0.90 (0.01)	4184.74 (86.19)	5.5 (0.97)
	0.8	0.86 (0.02)	0.91 (0.00)	4638.37 (118.37)	5.4 (1.35)
	1.0	**0.89 (0.01)**	0.93 (0.01)	5476.84 (155.83)	**4.3 (1.64)**
Mutagenesis	0.2	0.87 (0.05)	0.89 (0.01)	**504.84 (19.79)**	3 (0.67)
	0.4	0.83 (0.04)	0.87 (0.02)	519.97 (19.90)	**2.7 (0.82)**
	0.6	**0.88 (0.03)**	0.90 (0.01)	546.32 (22.35)	4.4 (0.84)
	0.8	**0.88 (0.01)**	0.94 (0.01)	558.25 (24.09)	4.61 (0.84)
	1.0	0.84 (0.03)	0.93 (0.01)	644.03 (26.77)	6.9 (1.59)
Pima-Indians	0.2	0.76 (0.01)	0.82 (0.01)	**1214.75 (31.86)**	**8.40 (1.84)**
	0.4	0.76 (0.02)	0.77 (0.01)	1234.85 (82.56)	10.9 (2.18)
	0.6	0.76 (0.01)	0.79 (0.01)	1401.14 (23.36)	14.7 (0.67)
	0.8	0.76 (0.02)	0.79 (0.01)	1628.93 (90.07)	13.4 (3.24)
	1.0	**0.77 (0.02)**	0.79 (0.01)	1697.19 (111.97)	11.5 (1.90)
Vote	0.2	**0.94 (0.04)**	0.95 (0.01)	**674.26 (38.01)**	**2.1 (0.32)**
	0.4	0.93 (0.05)	0.95 (0.01)	928.54 (31.95)	3.2 (1.23)
	0.6	**0.94 (0.03)**	0.95 (0.01)	1247.39 (51.21)	4.6 (2.01)
	0.8	**0.94 (0.02)**	0.95 (0.01)	1412.48 (35.54)	5.4 (1.51)
	1.0	0.92 (0.02)	0.95 (0.01)	1569.21 (78.03)	5.9 (1.52)

Table 7.7: Second set of experiments with different values of pbk. pbk is the probability that a fact of the background knowledge is selected at each iteration of ECL.

This is due to the cost of evaluating clauses. In fact, the derivation tree built by Progol in order to control if a clause covers an example or not, becomes bigger with higher values of pbk. This is because more background knowledge means more possible derivations that can be built for a given query to a logic program, as explained in chapter 2.

It is interesting to notice that for some datasets (Breast, Heart, Hepatitis, Ionosphere, Pima-Indians, Accidents) the simplicity increases with increasing values of pbk until $pbk \leq 0.8$, and it decreases when the whole background knowledge is used. In the other cases the simplicity increases when bigger portions of the background knowledge are used. This is because ECL achieves a higher diversity in the population, and the procedure for the extraction of the final solution adds more clauses to the final solution. In fact this procedure adds clauses to the emerging logic program as long as the accuracy of the logic program does not decrease.

The last thing that is interesting to notice is that for the Breast dataset, with

pbk set to 0.6, ECL obtains the best results, in a time comparable to the one required by the setting ECL-GA presented in section 7.2. With pbk set to 0.2, ECL could obtain an accuracy higher than the one obtained by both ECL-GA and ECL-NOT in a less amount of time.

Choosing the Value of pbk We propose here a simple procedure that can be used for choosing a good value of pbk. This procedure is not guaranteed to find the optimal value of pbk, it represents only a way for finding a good value of pbk. We suppose here that a k–fold cross validation is used. In the settings of parameters of an EA, it is important to recall that only the results on the training sets should be used, i.e., the test sets should not be accessed during the runs conducted for setting the parameters.

First a set of initial values of pbk that we want to test should be established. Let us denote this set as $\{pbk_1, pbk_2, \ldots, pbk_n\}$, where $pbk_i < pbk_{i+1}, 1 \leq i < n$. The procedure that can be used for establishing a good value of pbk is then the following:

Choose_pbk

1. $i = 1$, perform a run for each training fold with pbk set to pbk_i;
2. Perform a run for each training fold with pbk set to pbk_{i+1};
3. Register the average accuracies, and average simplicities obtained on the k training folds with pbk_i and pbk_{i+1} (let us denote this with acc_{pbk_j} and $simp_{pbk_j}$, respectively, $j \in \{i, i+1\}$);
4. If $acc_{pbk_i} > acc_{pbk_{i+1}}$ then go to step 7;
5. $i = i + 1$;
6. If $i < n$ go to step 2;
7. Choose pbk_j, $1 \leq j \leq i$, such that acc_{pbk_j} is the highest. If the highest value of acc_{pbk_i} was obtained with different pbk_i, then choose the pbk_i such that $simp_{pbk_i}$ is the lowest. If even the values of $simp_{pbk_i}$ are equal, then choose the smallest pbk_i.

For example, if we apply this procedure to the benchmark datasets used in the experiments presented in this section, we obtain the values of pbk given in table 7.8. In the third column we indicate if the suggested value is the one yielding the best accuracy on the test sets.

In six cases, the average accuracy obtained on the test sets achieved with the value of pbk found with the proposed procedure was not the highest. However, only on the Glass2 dataset the suggested value of pbk yields an average accuracy on the test sets that is evidently lower than the one obtained with the optimal value. In the other cases, the accuracy obtained on the test sets with the suggested values of pbk is only slightly worse than the one obtained with the optimal values of pbk. In six cases, the procedure finds the values of pbk

Dataset	pbk suggested	Winner	Difference
Accidents	1.0	Yes	0
Australian	0.4	Yes	0
Breast	0.8	No	0.01
Congestions	1.0	Yes	0
Crx	0.8	Yes	0
Echocardiogram	1.0	Yes	0
German	0.4	Yes	0
Glass2	1.0	No	0.05
Heart	1.0	No	0.01
Hepatitis	0.6	No	0.02
Ionosphere	0.2	Yes	0
Mutagenesis	0.2	No	0.01
Pima-Indians	0.2	No	0.01
Vote	0.2	Yes	0

Table 7.8: Values of pbk obtained with the procedure Choose_pbk. In the column labeled "Winner" we indicate if the suggested value is the winner on the test set. In the last column the difference between the best accuracy and the accuracy obtained with the suggested value of pbk is reported.

in the first two steps, requiring in this way little effort for setting the value of pbk. Among these six cases, the suggested value represents the optimal value in five cases, and only on the Mutagenesis dataset the suggested value does not yield the best performances on the test set.

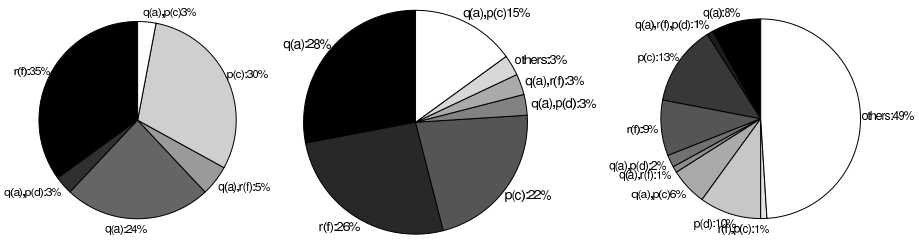
7.4 Experiments on the Selection Operators

In this section we perform some experiments in order to establish the beneficial properties of the variants of the US operator described in chapter 5. We first perform experiments on an artificially generated dataset, and then on real life datasets. The artificial dataset is made of five hundred positive examples and five hundred negative examples. Each example can be described by three attributes q, p and r, that can assume respectively the values $\{a, b\}$, $\{c, d, e\}$ and $\{f, g\}$.

Attr	a,c	a,d	a,e	b,c	b,d	b,e	f	g
Pos	0.50	0.30	0.05	0.05	0.05	0.05	0.70	0.30
Neg	0.05	0.05	0.50	0.20	0.10	0.10	0.30	0.70

Table 7.9: Probabilities of having a particular combination of attributes/values describing an example, given a positive (Pos) or a negative (Neg) example in the dataset.

The dataset was generated with the probabilities given in table 7.9. For instance, the probability of having a positive example described by the attribute q with value a and by the attribute p with value c is 0.5, while the probability of having a negative example described by the same pair of attributes values is 0.05.



(a) US (b) WUS (c) EWUS

Figure 7.5: Distribution of different rules in the final population obtained with the use of the different selection operators.

Figures 7.5(a), 7.5(b) and 7.5(c) show the percentage of different kind of clauses in the final population obtained with the use of the US, the WUS and the EWUS selection operator, respectively. The difference between the US and the EWUS selection operator is evident here. In the population obtained with the US selection operator, there are three kind of rules that dominate the population, namely $r(X, f), q(X, a)$ and $p(X, c)$. Instead, in the population obtained with the EWUS operator the distribution of different kind of rules is more homogeneous. There are no individuals that dominate the population. Also the WUS operator shows better results regarding the diversity in the population. There are still individuals that dominate the population, but the distribution of different kind of rules is more even.

	US	WUS	EWUS
Uncovered	13.5 (3.53)	13 (0.01)	0.67 (0.58)
Average	19.80 (0.24)	17.20 (3.11)	9.00 (0.23)

Table 7.10: The first row of the table shows how many positive examples were still uncovered after the GA ended. The second row shows the average of the dimension of the coverage sets, which is how many individuals cover a positive example.

Table 7.10 shows another result in which we were interested: how the positive examples are covered by population evolved by the system using the three selection operators. Again, results obtained with the EWUS operator are better than the ones obtained with the other two operators. Practically with the use of

Dataset	Mechanism	Diversity	Unc.	Cov.	Accuracy	Simplicity
Accidents	US	12.7 (3.12)	1.4 (0.70)	17.92 (4.00)	0.92 (0.01)	**3.2 (1.14)**
	WUS	11.2 (1.55)	1.2 (0.92)	17.73 (3.21)	0.94 (0.02)	3.3 (0.83)
	EWUS	**15.4 (1.78)**	**0.3 (0.48)**	**14.45 (3.48)**	**0.95 (0.02)**	3.5 (0.49)
Australian	US	11.9 (5.74)	10.5 (4.20)	38.54 (5.09)	**0.85 (0.04)**	**2.2 (1.23)**
	WUS	9.7 (3.20)	11.9 (3.84)	41.38 (1.91)	**0.85 (0.02)**	2.6 (0.97)
	EWUS	**29.7 (4.45)**	**1.8 (2.20)**	**24.52 (3.21)**	**0.85 (0.01)**	5.5 (1.43)
Breast	US	**19.1 (1.85)**	10.8 (5.71)	16.74 (2.03)	0.94 (0.03)	7.0 (1.05)
	WUS	18.6 (1.58)	10.5 (4.11)	15.59 (2.24)	0.94 (0.03)	**6.9 (1.59)**
	EWUS	17.8 (3.46)	**7.2 (5.12)**	**10.41 (1.88)**	**0.95 (0.02)**	8.6 (1.65)
Congestions	US	13.1 (2.84)	4.3 (2.17)	17.16 (4.33)	0.92 (0.02)	3.3 (0.82)
	WUS	12.8 (3.15)	2.0 (1.05)	15.32 (2.63)	0.93 (0.02)	**3.2 (1.03)**
	EWUS	**13.2 (2.44)**	**0.4 (0.70)**	**10.70 (2.52)**	**0.94 (0.02)**	3.5 (0.49)
Crx	US	3.2 (0.79)	21.5 (2.95)	70.49 (1.47)	**0.85 (0.04)**	**1.6 (0.70)**
	WUS	4.6 (1.51)	19.5 (6.47)	68.70 (2.57)	**0.85 (0.04)**	1.9 (0.99)
	EWUS	**28.1 (10.70)**	**4.8 (3.85)**	**29.24 (8.03)**	0.84 (0.01)	5.0 (1.76)
Echocardiogram	US	142.1 (19.34)	0.87 (0.03)	259.86 (17.40)	0.73 (0.03)	2.8 (0.79)
	WUS	140.3 (20.16)	**0 (0.0)**	259.14 (14.28)	**0.76 (0.03)**	**2.5 (0.71)**
	EWUS	**154.5 (15.64)**	**0 (0.0)**	**216.52 (23.28)**	0.74 (0.01)	3.0 (0.81)
German	US	49.7 (7.27)	1.2 (0.03)	166.22 (6.34)	**0.74 (0.03)**	**8.2 (2.10)**
	WUS	50.1 (8.45)	**0 (0.0)**	168.43 (7.00)	0.73 (0.04)	8.7 (2.21)
	EWUS	**71.9 (4.65)**	**0 (0.0)**	**144.20 (5.98)**	**0.74 (0.01)**	11.7 (0.24)
Glass2	US	104.3 (24.21)	0.6 (0.04)	403.49 (4.48)	0.84 (0.02)	**2.7 (0.67)**
	WUS	123.8 (8.73)	**0 (0.0)**	399.45 (4.65)	0.83 (0.02)	**2.7 (0.95)**
	EWUS	**180.4 (30.00)**	**0 (0.0)**	**356.14 (19.50)**	**0.85 (0.01)**	3.7 (1.06)
Heart	US	11.4 (2.01)	7.1 (3.45)	32.85 (1.84)	0.80 (0.05)	3 (0.94)
	WUS	11.6 (4.35)	5.1 (3.07)	31.78 (3.27)	0.76 (0.04)	**2.7 (0.67)**
	EWUS	**41.7 (3.20)**	**0.9 (0.57)**	**28.57 (3.11)**	**0.81 (0.03)**	2.9 (0.74)
Hepatitis	US	43.7 (5.82)	1.23 (0.02)	229.56 (3.57)	0.80 (0.03)	**5.6 (1.17)**
	WUS	45.3 (4.99)	**0 (0.0)**	230.20 (3.56)	0.81 (0.04)	6.7 (1.16)
	EWUS	**58.0 (6.50)**	**0 (0.0)**	**221.31 (2.62)**	**0.83 (0.02)**	6.3 (1.25)
Ionosphere	US	126.6 (8.64)	1.2 (0.18)	**160.42 (2.74)**	0.87 (0.02)	**8.0 (2.36)**
	WUS	122.9 (9.69)	**0 (0.0)**	260.93 (11.73)	0.88 (0.03)	8.4 (1.35)
	EWUS	**195.0 (6.32)**	**0 (0.0)**	222.07 (9.60)	**0.89 (0.02)**	12.0 (1.76)
Mutagenesis	US	29.6 (5.87)	1.2 (0.43)	119.66 (5.13)	0.87 (0.02)	3.1 (0.74)
	WUS	30.0 (5.696)	**0 (0.0)**	120.281 (4.07)	**0.90 (0.02)**	**2.9 (0.99)**
	EWUS	**41.8 (7.60)**	**0 (0.0)**	**109.86 (7.18)**	0.88 (0.01)	4.6 (0.84)
Pima-Indians	US	**22.00 (2.36)**	**0.0 (0.0)**	58.71 (2.10)	0.75 (0.02)	**7.5 (1.65)**
	WUS	21.6 (1.78)	**0.0 (0.0)**	57.74 (2.85)	0.74 (0.05)	8.1 (2.23)
	EWUS	20.3 (1.34)	**0.0 (0.0)**	**51.37 (2.25)**	**0.77 (0.02)**	7.9 (1.29)
Vote	US	29.4 (3.53)	1.1 (0.32)	129.82 (5.73)	0.92 (0.04)	**3.1 (1.20)**
	WUS	30.0 (4.42)	0.2 (0.42)	132.39 (3.87)	0.93 (0.04)	3.4 (1.51)
	EWUS	**48.0 (9.61)**	**0 (0.0)**	**127.38 (2.18)**	**0.94 (0.02)**	3.7 (1.06)

Table 7.11: Results for the various selection mechanisms. The column labeled Unc. report the average number of positive examples that are not covered by any individual in the evolved population. The column labeled Cov. reports the average number of individuals that cover a positive example. Standard deviation is reported between brackets. Best results are highlighted.

Dataset	US	WUS	EWUS
Accidents	E	E	
Australian	E	E	
Breast			
Congestions			
Crx	W,E	E	
Echocardiogram	(E)	(E)	
German	E	E	
Glass2	(W),E	E	
Heart	E	E	
Hepatitis	E	E	
Ionosphere	E	E	
Mutagenesis	E	E	
Pima-Indians			
Vote	E	E	

Table 7.12: Results of t-test for the diversity of the final population. For each dataset we report when a method achieved significant better result than another, as estimated by t-test with confidence level of 1%, where E stands for EWUS, W for WUS, U for US. A symbol between brackets stands for a confidence level of 5%.

the EWUS operator, all the positive examples are covered, while with the other two variants thirteen positive examples are still uncovered after the GA has ended. This is due to the more variety in the population evolved by the system with the use of the EWUS selection operator. The second row of table 7.10 also indicates that the population obtained with the EWUS operator is more spread through the hypothesis space than the one obtained with the other two operators. For this result the WUS operator performed a little bit better than the US operator, however these results are significantly worse than those obtained using the EWUS operator.

Table 7.11 contains results on the considered benchmark datasets. It can be seen that in most of the cases, the EWUS selection operator leads to a better diversity in the evolved final population. Only in two cases (the Breast and the Pima-Indians datasets) the population evolved with the standard US operator has a slightly higher diversity. The WUS selection operator was less successful than the EWUS in promoting diversity. In most of the cases the diversity obtained by the WUS operator is comparable to the diversity obtained by the US operator. In table 7.12, we further analyze the results regarding the diversity achieved with the use of the three selection operators. In the table we report the results of the statistical paired two–tailed t-test regarding the diversity with confidence level 1% and 5%. An entry of the table contains a symbol relative to a selection operator, if the indicated selection operator achieved a significantly higher diversity than the one achieved by the selection operator relative

to the column for the particular dataset relative to the row. We can notice that EWUS was never outperformed by the other selection mechanisms. Using a confidence level of 1% we can see that EWUS outperformed both US and WUS on the same ten datasets. WUS outperformed once, on the Crx dataset, the US selection operator, while US never outperformed the other selection mechanisms. If we extend the confidence level to 5%, then we can see that EWUS outperformed the US and the WUS also on the Echocardiogram dataset, and the WUS outperformed the US also on the Glass2 dataset.

Generally, there are also less uncovered positive examples at the end of the evolution when EWUS is used. This also confirms the fact that the population evolved by ECL with EWUS is more spread out through the hypothesis space than the population evolved with the use of the other two selection mechanisms. The fifth column presents the average number of individuals in the final population covering a positive example. Also these results confirm the effectiveness of the EWUS selection operator in promoting diversity. Having higher diversity implies having less individuals covering the same examples. Higher diversity in the population is also positively reflected in the quality of the solution found. In fact the solutions found with the use of EWUS generally have a higher accuracy than the solutions found with the other two selection mechanisms.

In almost all the cases the solutions found by ECL with the use of the EWUS operator are less simple than those found with the use of the other selection operators. In some cases, like in the Ionosphere, the Crx and the Australian datasets, the difference is evident. This is due to the fact that the population evolved with the use of EWUS is characterized by a higher diversity. More diversity in the population means that more rules are taken in consideration for being added to the final solution, thus the solution extracted can contain more rules. This can be noticed for example in the Ionosphere, the Crx and the Australian cases, where the solutions found are more complex and where the population evolved with the EWUS operator is much more diverse than the other two populations. However this cannot be considered as a negative fact. In fact, in the other cases, the difference in simplicity is not evident, and in many cases is similar to the simplicity of solutions found with US and WUS.

7.5 Experiments on Solution Extraction

In this section we experimentally compare the three methods that can be used for the extraction of the final solution, and we identify them as follows:

WSCAf is the method based on the heuristic for solving weighted set covering problems using the weights given by equation 5.4;

WSCAn is the method relying on the same heuristic but using weights given by equation 5.5;

Precision is the procedure based on precision, given in figure 5.5.

The three methods are described in chapter 5.

Table 7.13 reports the results obtained. The table shows the average accuracies and the average simplicity of the solution extracted when the three methods are applied to the same final population. In the table, a • next to a result, stands for a significant difference between the accuracy obtained by the relative method and the accuracy obtained with Precision as estimated by the statistical paired two-tailed t-test, with confidence level 1%. A (•) stands for a significant difference with confidence level 5%. We performed the t-test in the same way as in section 7.2. The setting given in section 7.1 was used in all the experiments. The only difference between the obtained results lies in the method applied for the extraction of the final solution.

From the results, it emerges that the procedure based on precision can always find a solution of higher accuracy. In particular, for the Australian, the Glass2, the Heart, and the Congestions dataset the difference in the accuracy of the solutions extracted is evident. It should also be noticed that in the cases where the t-test is applicable, only on the Breast dataset the difference in the obtained results is not significant. On the Ionosphere dataset only the results obtained by WSCAf are significantly worse than the results obtained by Precision. In all the other cases, the difference is significant. The t-test is not applicable on the Echocardiogram, the Glass2, the Heart and the Hepatitis datasets, since results on these datasets are not normally distributed.

Generally, the accuracy of the solutions extracted by WSCAn is slightly better than the accuracy of the solution extracted by WSCAf. This is particularly true for the Australian, the Crx, the Glass2, the Heart, the Ionosphere and the Congestions datasets. However, these differences are not significant. In the cases where the accuracy of the solution found by WSCAn and WSCAf is equal, many of the solutions extracted with the two methods are formed by exactly the same logic programs.

The solutions extracted by both WSCAn and WSCAf are simpler than those extracted by Precision. However since the difference in the accuracy is so remarkable, we cannot conclude that this is a positive effect. The solutions found by the procedure based on precision are in some cases much more complex, e.g., in the Ionosphere case, but the accuracy of the solution is higher. The reason why solutions extracted by Precision contain more clauses is because with this method the most precise clause is always added to the emerging solution, as long as the accuracy of the solution does not decrease. In this way, many very specific clauses can be added. This is less likely to happen in both WSCAn and in WSCAf. In fact these two methods try to cover all the positive examples with a minimum weight. For this reason it is likely that the final solution consists of less clauses.

We can conclude that, as stated in chapter 5, in many cases, the procedure based on the weighted covering set algorithm is not capable of extracting a solution of good quality, even when the clauses for building an accurate logic program are present in the population. This is demonstrated by the fact the the procedure based on the precision is able, given the same set of clauses, to extract a solution of satisfying quality for all the cases.

Dataset	Mechanism	Accuracy	T-test	Simplicity
Accidents	WSCAf	0.92 (0.01)	•	**2.30 (0.48)**
	WSCAn	0.93 (0.01)	(•)	2.70 (0.67)
	Precision	**0.94 (0.02)**		3.70 (1.06)
Australian	WSCAf	0.62 (0.04)	•	**5.30 (1.16)**
	WSCAn	0.67 (0.02)	•	5.70 (1.34)
	Precision	**0.85 (0.01)**		6.10 (2.18)
Breast	WSCAf	0.94 (0.01)		**5.80 (1.40)**
	WSCAn	0.94 (0.02)		6.10 (1.45)
	Precision	**0.95 (0.02)**		8.60 (0.41)
Congestions	WSCAf	0.85 (0.03)	•	**3.70 (1.16)**
	WSCAn	0.90 (0.02)	(•)	4.20 (1.55)
	Precision	**0.94 (0.02)**		3.95 (0.35)
Crx	WSCAf	0.64 (0.01)	•	4.90 (0.74)
	WSCAn	0.67 (0.02)	•	5.40 (1.43)
	Precision	**0.84 (0.01)**		**4.80 (0.05)**
Echocardiogram	WSCAf	0.60 (0.03)		3.30 (0.48)
	WSCAn	0.60 (0.02)	NA	3.20 (0.42)
	Precision	**0.74 (0.01)**		**2.60 (0.70)**
German	WSCAf	0.70 (0.01)	•	**5.30 (2.71)**
	WSCAn	0.71 (0.01)	•	**5.30 (2.79)**
	Precision	**0.74 (0.01)**		11.70 (0.24)
Glass2	WSCAf	0.73 (0.03)		3.50 (0.71)
	WSCAn	0.75 (0.02)	NA	**3.30 (0.48)**
	Precision	**0.85 (0.01)**		4.20 (1.23)
Heart	WSCAf	0.60 (0.03)		4.50 (0.53)
	WSCAn	0.64 (0.02)	NA	4.30 (0.67)
	Precision	**0.80 (0.03)**		**4.20 (1.32)**
Hepatitis	WSCAf	0.82 (0.01)		4.20 (0.63)
	WSCAn	0.82 (0.02)	NA	**4.10 (0.94)**
	Precision	**0.83 (0.02)**		7.60 (0.95)
Ionosphere	WSCAf	0.81 (0.03)	•	**2.90 (0.32)**
	WSCAn	0.87 (0.03)		3.80 (1.99)
	Precision	**0.89 (0.02)**		12.50 (1.48)
Mutagenesis	WSCAf	0.77 (0.04)	•	3.12 (0.32)
	WSCAn	0.80 (0.03)	(•)	**2.83 (0.63)**
	Precision	**0.88 (0.01)**		4.61 (0.84)
Pima-Indians	WSCAf	0.71 (0.02)	(•)	**6.10 (0.74)**
	WSCAn	0.71 (0.02)	(•)	**6.10 (0.72)**
	Precision	**0.76 (0.01)**		8.40 (1.84)
Vote	WSCAf	0.83 (0.04)	(•)	4.30 (0.48)
	WSCAn	0.85 (0.02)	(•)	4.50 (0.53)
	Precision	**0.94 (0.02)**		**3.70 (1.06)**

Table 7.13: Average accuracies and average simplicities obtained with the three methods for the extraction of the final solution. Standard deviation between brackets. A • stands for a significant difference related to accuracy as estimated by t-test. NA means that the t-test is not applicable.

7.6 Experiments on Discretization Methods

The experiments presented in this section are aimed at evaluating the discretization methods described in chapter 6:

ECL-LSDc with local supervised discretization and a coarse initialization of inequalities using DP points;

ECL-LSDf as the previous variant but with a fine initialization of inequalities using BP points;

ECL-LUD with local unsupervised discretization;

ECL-GSD where Fayyad & Irani discretization algorithm is applied prior to induction.

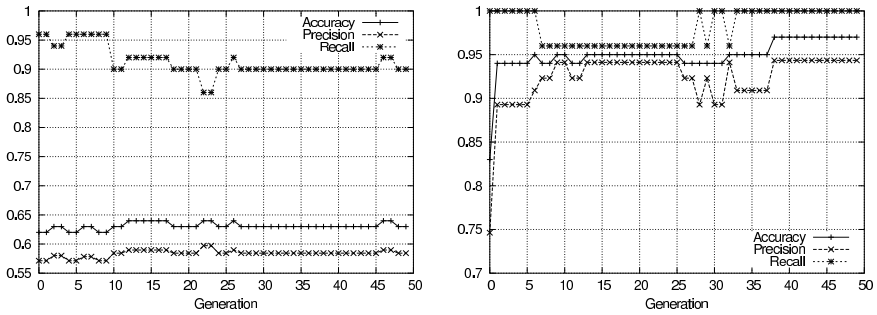
First, we use the artificial dataset discussed in the chapter 6 and illustrated in figure 6.1, for analyzing the behavior ECL when the four discretization methods are embedded into the system and the resulting four variants of ECL are applied to this non-linearly separable problem. Next, we consider real-life learning tasks and perform experiments on propositional and relational datasets, chosen for the high presence of numeric values. In ECL-LUD we have used the WEKA implementation (Witten and Frank, 2000b) of the EM algorithm, in order to perform clustering of numerical values.

7.6.1 Artificially Generated Dataset

In this experiment, 50 positive and 50 negative examples of the target concept described in section 6.1 are fed to the system. Each example is described by two numerical attributes that can assume values in $[0, 1000]$. The system is run with population size equal to 100, for 50 generations and with 30 individuals selected at each generation, while the greediness N_i of the mutation operators and the maximum length of a clause lc are set to 3.

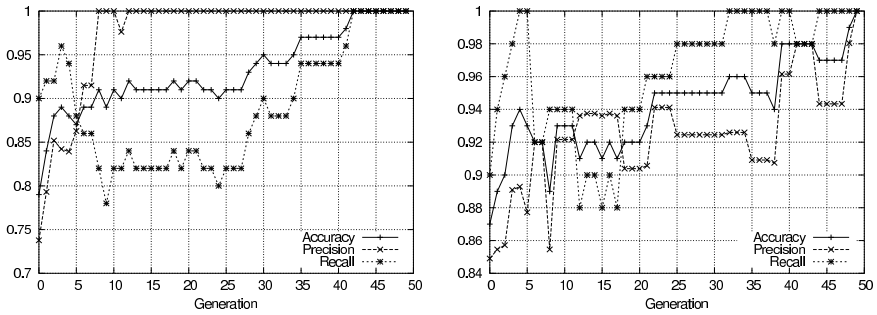
ECL-GSD is not able to solve this problem because no discretization point is found by the Fayyad & Irani's method. Thus both continuous attributes are discretized in a unique interval by the Fayyad & Irani's algorithm. For the same reason, ECL-LSDc has low performance, as shown in figure 7.6(a) where accuracy, precision and recall of the extracted solution are plotted at each generation of a typical run, indicating that there is no evolution.

The other two ECL variants, ECL-LUD and ECL-LSDf, have satisfactory performance. Figure 7.6(b) shows a typical run of ECL-LUD, where the accuracy of the extracted solution can be seen to increase, even if rather slowly, during the generations, and it becomes constant after about 40 generations, where recall becomes equal to 1 and precision is about 0.95. Figure 7.7 shows accuracy, precision and recall of the solution extracted at each of the 50 generations in two typical runs of ECL-LSDf. It can be seen that after some oscillations a perfect solution is found, like the one consisting of the following two clauses:



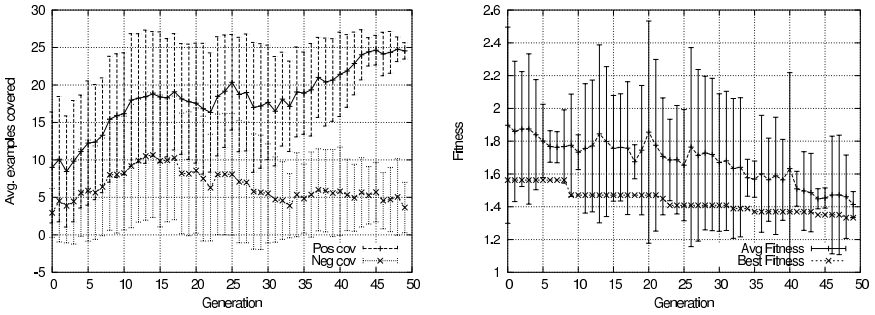
(a) ECL-LSDc. (b) ECL-LUD.

Figure 7.6: Average accuracy, precision and recall of the solution found at every generation of a typical run of ECL-LSDc and ECL-LUD.



(a) Run I. (b) Run II.

Figure 7.7: Average accuracy, precision and recall of the solution extracted at every generation of ECL-LSDf during two typical runs on the artificial dataset.



(a) Average coverage of individuals.

(b) Average and best fitness of individuals.

Figure 7.8: Graphs for a typical run of ECL-LSDf. Vertical bars show standard deviation.

$$\texttt{case(X)} : -\texttt{attr}_1\texttt{(X,Y)}, \texttt{attr}_2\texttt{(X,Z)}, (611.56 < \texttt{Z} \leq 976.55), \\ (516.79 < \texttt{Y} \leq 975.36).$$
$$\texttt{case(X)} : -\ \texttt{attr}_1\texttt{(X,Y)}, \texttt{attr}_2\texttt{(X,Z)}, (-\infty < \texttt{Z} \leq 487.58), (21.24 < \texttt{Y} \leq 489.59).$$

Figure 7.8(a) shows the average number of positive and negative examples covered by the individuals of the population at every generation, and figure 7.8(b) the average and best fitness at each generation. The fine initialization of inequalities with BP intervals allows the algorithm to progressively enlarge the boundaries of inequalities and correctly classify more and more examples until a solution is found. The graphs shown in this section are obtained from typical runs. The same graphs relative to the average values obtained on a number of runs, present the same behavior, so the same conclusions can be drawn from them.

Thus ECL-LSDf seems the best choice for handling this type of problems. However, we will see in the next sections that on real-life datasets ECL-LSDc yields the best accuracy.

7.6.2 Propositional Datasets

In this section we test the four ECL variants on the ten propositional datasets described in table 7.1.

Table 7.14 contains the total number of DP and BP points of the datasets, showing that in general the latter is much bigger than the former. Tables 7.15 and 7.16 contain the results of the experiments on the training and test sets, respectively. On the training sets ECL-LSDf obtains the best performance on all datasets, with optimal performance on the Echocardiogram. However, on the test sets, ECL-LSDc achieves the best accuracy in most of the cases, with simplicity (that is, the number of clauses of the output program) that is second

Dataset	DP	BP
Australian	13	810
Breast	29	84
Crx	13	806
Echocardiogram	5	151
German	30	256
Glass2	16	479
Heart	10	294
Hepatitis	10	192
Ionosphere	145	1359
Pima-Indians	17	1123
Liver	7	275
Sonar	81	5965
Wdbc	91	5184
Wpbc	34	2313

Table 7.14: Total number of DP and BP points per propositional dataset.

best after ECL-GSD. ECL-LSDf produces best results on the Echocardiogram and Hepatitis dataset, ECL-GSD on Glass2, but the results are only slightly better than those of ECL-LSDc. The unsupervised variant ECL-LUD produces satisfactory approximate solutions, yet of quality inferior to that of the other methods. The training time of the four algorithms is comparable, where ECL-LSDc and ECL-GSD are slightly faster than the other variants.

In order to summarize the performance of the four variants and the significance of the results with respect to the accuracy, we compute the statistical paired two-tailed t-test with confidence level of 1% and 5%. The t-test is performed on the 30 results obtained from the 10 folds and the 3 random seeds. In table 7.18 we report the results of the t-test for each of the propositional datasets. In the columns we report the discretization methods, and in the rows the dataset. We report in an entry of the table the symbol associated to another discretization method if the discretization method identified by the symbol is superior to the discretization method relative to the column on the dataset relative to the row. For example, ECL-LSDc is better than all the other methods on the Pima-Indians dataset. If the symbol is between brackets it means that the confidence level is 5%. Table 7.19 summaries the results reported in table 7.18. From the table we can extract the following hierarchy for the methods: ECL-LSDc, ECL-LSDf, ECL-GSD, and ECL-LUD.

Using 1% confidence level we get that ECL-LSDc is never outperformed, while it is significantly better than the other methods on the Pima-Indians dataset, better than ECL-GSD on the Breast and on the Wdbc datasets, and better than ECL-LUD on the Sonar and the Ionosphere datasets, together with ECL-GSD. On the Ionosphere dataset also ECL-LSDf is better than ECL-LUD. ECL-LSDc is significantly better than ECL-LSDf also on the Liver dataset.

Dataset	ECL-LSDc	ECL-LSDf	ECL-LUD	ECL-GSD
Australian	0.85 (0.02)	0.85 (0.03)	0.86 (0.01)	0.84 (0.01)
Breast	0.96 (0.00)	0.96 (0.00)	0.96 (0.01)	0.94 (0.01)
Crx	0.86 (0.01)	0.87 (0.01)	0.87 (0.01)	0.85 (0.01)
Echocardiogram	0.82 (0.01)	1.00 (0.00)	1.00 (0.01)	0.69 (0.02)
German	0.76 (0.00)	0.78 (0.00)	0.77 (0.01)	0.75 (0.00)
Glass2	0.86 (0.01)	0.95 (0.02)	0.95 (0.01)	0.87 (0.01)
Heart	0.83 (0.03)	0.83 (0.03)	0.84 (0.02)	0.83 (0.02)
Hepatitis	0.88 (0.01)	0.90 (0.01)	0.90 (0.01)	0.87 (0.02)
Ionosphere	0.93 (0.02)	0.97 (0.01)	0.93 (0.01)	0.91 (0.02)
Pima-Indians	0.77 (0.01)	0.82 (0.01)	0.79 (0.01)	0.69 (0.03)
Liver	0.70 (0.02)	0.71 (0.01)	0.69 (0.02)	0.63 (0.01)
Sonar	0.81 (0.02)	0.86 (0.01)	0.83 (0.04)	0.78 (0.02)
Wdbc	0.95 (0.01)	0.96 (0.01)	0.94 (0.01)	0.94 (0.01)
Wpbc	0.79 (0.01)	0.90 (0.02)	0.89 (0.01)	0.76 (0.01)

Table 7.15: Results of 3 runs with different random seeds and ten–fold cross validation on the training sets: average accuracy with standard deviation between brackets.

If we increase the confidence level to 5% then we get that ECL-LUD and ECL-LSDc are significantly better than ECL-LSDf on the Australian dataset, ECL-LSDf becomes also significantly better than ECL-GSD on the Breast dataset, and ECL-LSDc (together with ECL-GSD) becomes significantly better than ECL-LSDf and ECL-LUD on the German and on the Sonar dataset. ECL-LSDc becomes better than ECL-LUD also on the Wdbc dataset, and better than ECL-GSD on the Liver dataset. The other datasets (Echocardiogram, Glass 2, Heart, and Hepatitis) are small, and the results of the experiments are not normally distributed, so the t-test cannot be applied.

In general, simple solutions are obtained using Fayyad & Irani's discretization applied either prior to induction (ECL-GSD) or in the initialization of the inequalities (ECL-LSDc). The simplicity column of the results also indicates that the solutions produced by ECL-LSDf are in general more complex than those generated by the other methods, due to the initialization of the inequalities to rather small intervals.

In summary, the results of the experiments on these propositional datasets seem to indicate that an effective search strategy for discretizing continuous attributes in an evolutionary learner consists of starting from large intervals for initializing inequalities and then refine them during the evolutionary process using the boundary points for enlarging and shrinking the intervals. The results also indicate that the supervised methods obtain in general better performance than the unsupervised one. For this reason we will not consider ECL-LUD in the experiments on relational datasets described in the next section.

Dataset	System	Accuracy	Simplicity	Time (s)
Australian	ECL-LSDc	**0.85 (0.01)**	6.10 (2.18)	**1686.38 (144.07)**
	ECL-LSDf	0.83 (0.01)	15.50 (3.69)	2088.13 (114.96)
	ECL-LUD	**0.85 (0.01)**	16.6 (2.72)	1798.38 (55.13)
	ECL-GSD	0.84 (0.01)	**3.20 (0.79)**	2042.38 (342.80)
Breast	ECL-LSDc	**0.95 (0.02)**	8.60 (0.41)	286.13 (37.00)
	ECL-LSDf	0.94(0.03)	13.75 (2.05)	299.63 (27.21)
	ECL-LUD	0.94 (0.02)	14.10 (2.08)	521.50 (41.43)
	ECL-GSD	0.93 (0.02)	**6.05 (1.34)**	**274.23 (31.54)**
Crx	ECL-LSDc	**0.84 (0.01)**	4.80 (0.05)	**2668.00 (176.45)**
	ECL-LSDf	0.82 (0.02)	11.90 (3.48)	2763.23 (121.47)
	ECL-LUD	0.83 (0.02)	9.80 (3.16)	2983.83 (138.87)
	ECL-GSD	0.83 (0.01)	**3.70 (0.83)**	2693.32 (167.63)
Echocardiogram	ECL-LSDc	0.74 (0.01)	2.60 (0.70)	2375.63 (126.62)
	ECL-LSDf	**0.76 (0.02)**	10.00 (1.15)	2383.01 (121.43)
	ECL-LUD	0.65 (0.01)	11.90 (2.02)	2507.82 (103.32)
	ECL-GSD	0.69 (0.02)	**1.30 (0.48)**	**2205.75 (119.39)**
German	ECL-LSDc	**0.74 (0.01)**	11.70 (0.24)	4605.75 (155.34)
	ECL-LSDf	0.72 (0.02)	58.40 (2.40)	5158.38 (263.62)
	ECL-LUD	0.71 (0.01)	80.30 (6.46)	5277.43 (174.27)
	ECL-GSD	**0.74 (0.01)**	**5.6 (0.57)**	**4597.27 (143.07)**
Glass2	ECL-LSDc	0.85 (0.01)	4.2 (1.23)	**1246.00 (55.94)**
	ECL-LSDf	0.75 (0.03)	21.8 (2.66)	1673.00 (96.12)
	ECL-LUD	0.71 (0.03)	24.40 (3.89)	2846.00 (142.05)
	ECL-GSD	**0.86 (0.02)**	**2.2 (0.42)**	1453.25 (132.66)
Heart	ECL-LSDc	**0.80 (0.03)**	4.20 (1.32)	436.38 (57.59)
	ECL-LSDf	0.73 (0.01)	9.20 (3.05)	516.50 (45.39)
	ECL-LUD	0.77 (0.02)	10.90 (2.73)	850.75 (138.40)
	ECL-GSD	0.77 (0.02)	**2.50 (0.71)**	**403.64 (48.72)**
Hepatitis	ECL-LSDc	0.83 (0.02)	7.60 (0.95)	**1056.73 (63.84)**
	ECL-LSDf	**0.84 (0.04)**	17.70 (2.15)	1165.88 (45.80)
	ECL-LUD	0.80 (0.03)	24.50 (4.06)	1686.63 (62.86)
	ECL-GSD	0.83 (0.03)	**6.40 (1.20)**	1194.75 (70.26)
Ionosphere	ECL-LSDc	**0.89 (0.02)**	12.50 (1.48)	5276.83 (138.93)
	ECL-LSDf	0.88 (0.04)	45.78 (5.37)	5285.61 (174.61)
	ECL-LUD	0.78 (0.02)	88.20 (11.25)	5991.42 (182.02)
	ECL-GSD	0.87 (0.03)	**9.80 (1.34)**	**4972.87 (125.83)**
Liver	ECL-LSDc	**0.67 (0.03)**	**1.1 (0.32)**	**75.10 (3.44)**
	ECL-LSDf	0.54 (0.04)	20.7 (1.70)	107.38 (3.00)
	ECL-LUD	0.56 (0.05)	19.7 (1.83)	196.75 (11.74)
	ECL-GSD	0.63 (0.05)	1.2 (0.43)	187.37 (16.20)

Table 7.16: Results for the various methods on the propositional datasets: average accuracy on the test sets, number of clauses (simplicity), and training time in seconds, with standard deviation between brackets.

Dataset	System	Accuracy	Simplicity	Time (s)
Pima-Indians	ECL-LSDc	**0.76 (0.01)**	8.40 (1.84)	1314.75 (31.86)
	ECL-LSDf	0.71 (0.01)	75.60 (5.23)	1328.32 (29.35)
	ECL-LUD	0.70 (0.01)	53.80 (6.66)	2920.00 (57.15)
	ECL-GSD	0.68 (0.02)	**2.20 (0.63)**	**1284.73 (38.74)**
Sonar	ECL-LSDc	**0.76 (0.03)**	2.9 (0.74)	**713.28 (54.86)**
	ECL-LSDf	0.64 (0.04)	30.0 (3.59)	1182.26 (47.92)
	ECL-LUD	0.58 (0.03)	32.2 (8.88)	1232.73 (35.71)
	ECL-GSD	0.75 (0.02)	2.4 (0.52)	716,93 (64.39)
Wdbc	ECL-LSDc	**0.95 (0.03)**	7.8 (2.15)	**1647.65 (113.37)**
	ECL-LSDf	0.91 (0.04)	15.5 (5.70)	2560.44 (120.904)
	ECL-LUD	0.90 (0.03)	24.2 (6.37)	1725.51 (97.12)
	ECL-GSD	0.91 (0.03)	**4.6 (1.43)**	2141.81 (116.00)
Wpbc	ECL-LSDc	**0.78 (0.04)**	**1.8 (0.92)**	180.082 (20.56)
	ECL-LSDf	0.74 (0.02)	19.9 (1.59)	243.70 (12.19)
	ECL-LUD	0.76 (0.03)	20.9 (0.99)	249.692 (7.17)
	ECL-GSD	0.76 (0.04)	2.1 (0.31)	**178.32 (13.42)**

Table 7.17: Second set of results for the various methods on the propositional datasets: average accuracy on the test sets, number of clauses (simplicity), and training time in seconds, with standard deviation between brackets.

Dataset	ECL-LSDc	ECL-LSDf	ECL-LUD	ECL-GSD
Australian		(•) (⋄)		
Breast				• (○)
Crx				
Echocardiogram				
German		(•) (⋆)	(•) (⋆)	
Glass2				
Heart				
Hepatitis				
Ionosphere			• ○ ⋆	
Pima-Indians		•	•	•
Liver		•		(•)
Sonar		(•) (⋆)	• ⋆	
Wdbc			(•)	•
Wpbc				

Table 7.18: Results of t-test for the propositional datasets. For each dataset we report when a method is significantly better than another, as estimated by t-test with confidence level of 1%, where • stands for ECL-LSDc, ○ for ECL-LSDf, ⋄ for ECL-LUD and ⋆ stands for ECL-GSD. A symbol between brackets stands for a confidence level of 5%.

Method	ECL-LSDc	ECL-LSDf	ECL-LUD	ECL-GSD	Total
ECL-LSDc	–	2(5)	3(5)	3 (4)	8 (14)
ECL-LSDf	0	–	1	0 (1)	1 (2)
ECL-LUD	0	0 (1)	–	0	0 (1)
ECL-GSD	0	0 (2)	2(3)	–	2 (5)
Total	0	2 (8)	6(9)	3(5)	

Table 7.19: Results of the two-tailed paired t-test for the propositional datasets with 1% confidence level: each entry contains the number of datasets on which the algorithm in the row is significantly better than the one in the column. The results of the test using 5% confidence level are reported between brackets when they differ from those using 1% confidence level.

7.6.3 Relational Datasets

The relational datasets subject of the experiments presented here, are used as benchmark problems for ILP systems (see, e.g., (Van Laer, 2002)). Table 7.20 contains the total number of DP and BP points per dataset.

Dataset	DP	BP
Mutagenesis	9	116
Accidents	7	121
Congestions	7	118
Bio-Fast	4	190
Bio-Slow	2	257
Bio-Moderate	2	311
Bio-Resistant	4	100

Table 7.20: Total number of DP and BP points per dataset.

We here consider each class of the traffic and the biodegradability datasets as a separate learning task, thus obtaining a total of seven binary classification problems. In section 7.7 both problems are considered as two multiclass problems. In chapter 9 we propose a detailed study of the traffic dataset.

Table 7.21 shows the average accuracies on the training sets and Table 7.22 the results on the test sets.

On the training set ECL-LSDc yields the best average accuracy on the first three datasets, ECL-LSDf outperforms the other algorithms on the last four datasets, and ECL-GSD yields reasonable results, slightly inferior to those of ECL-LSDc. The training time of the algorithms is comparable. However, the higher complexity of the solutions found by ECL-LSDf, containing on average twice the number of clauses of the other algorithms, penalizes its performance on the test sets.

On the mutagenesis dataset the accuracies obtained by the three systems are

Dataset	ECL-LSDc	ECL-LSDf	ECL-GSD
Mutagenesis	0.94 (0.01)	0.91 (0.02)	0.87 (0.03)
Accidents	0.95 (0.02)	0.94 (0.01)	0.93 (0.01)
Congestions	0.96 (0.01)	0.90 (0.01)	0.95 (0.00)
Bio-Fast	0.88 (0.01)	0.92 (0.00)	0.88 (0.01)
Bio-Slow	0.80 (0.01)	0.86 (0.00)	0.80 (0.01)
Bio-Moderate	0.75 (0.01)	0.80 (0.01)	0.75 (0.00)
Bio-Resistant	0.93 (0.01)	0.96 (0.00)	0.93 (0.01)

Table 7.21: Experiments on the relational datasets: average accuracies on training sets with standard deviation between brackets.

comparable, with ECL-LSDf performing slightly better than ECL-LSDc and ECL-GSD.

On the traffic dataset the best results are produced by ECL-LSDc, while ECL-LSDf obtains the worst performance. In (Džeroski et al., 1998a; Džeroski et al., 1998b) a discretization provided by experts in the field was used for the three numerical arguments of the traffic dataset. Using the same discretization, ECL-GSD obtained results that are slightly superior to those obtained using Fayyad & Irani's algorithm (on the Accidents dataset the average accuracy on the test and training sets is 0.92 (0.03) and 0.94 (0.02) and the average simplicity is 5.10 (0.93). On the Congestions dataset the average accuracy on the test and training sets is 0.93 (0.02) and 0.95 (0.00) and the average simplicity is 3.23 (0.21)). The two discretizations produce similar partitions for two of the three attributes.

On the biodegradability dataset ECL-LSDc obtains the best results on two of the three binary classification problems, and has slightly inferior performance on the Bio-Slow class.

Like for the propositional datasets, we analyze further the accuracy results by means of the statistical paired two-tailed t-test with 1% and 5% confidence levels. The results of the t-test are reported per dataset in table 7.23, and the summary of these results are reported in table 7.24. Also in this case, ECL-LSDc turns out to be the best algorithm. ECL-LSDc is never outperformed, and using 1% confidence level it is significantly better than ECL-LSDf on three datasets (Accidents, Congestion and Bio-Fast), and significantly better than ECL-GSD on two datasets (Bio-Moderate, Bio-Resistant). Moreover, ECL-GSD is significantly better than ECL-LSDf on two datasets (Bio-Fast, Congestions) while it is outperformed by ECL-LSDf on the Bio-Slow dataset. If we use a 5% confidence level, then we have that ECL-LSDc is better than ECL-LSDf also on the Bio-Moderate dataset, and better than ECL-GSD on the Accidents dataset.

In summary, for the relational datasets we can draw the same conclusions as for the propositional ones, namely that a good performance in terms of accuracy and simplicity is obtained by embedding in ECL a discretization method which initializes inequalities using Fayyad & Irani's algorithm, and then re-

Dataset	System	Accuracy	Simplicity	Time (s)
Mutagenesis	ECL-LSDc	0.88 (0.01)	4.61 (0.84)	558.25 (24.09)
	ECL-LSDf	**0.90 (0.01)**	7.92 (1.51)	**542.88 (27.88)**
	ECL-GSD	0.89 (0.01)	**2.71 (0.38)**	693.13 (35.71)
Accidents	ECL-LSDc	**0.95 (0.02)**	**3.55 (0.49)**	1083.8 (86.82)
	ECL-LSDf	0.87 (0.02)	15.55 (1.06)	1182.84 (73.83)
	ECL-GSD	0.92 (0.03)	5.10 (0.93)	**974.31 (80.28)**
Congestions	ECL-LSDc	**0.94 (0.02)**	3.95 (0.35)	**1107.30 (138.73)**
	ECL-LSDf	0.84 (0.01)	7.20 (0.57)	1489.51 (174.94)
	ECL-GSD	0.93 (0.02)	**3.23 (0.21)**	1145.87 (108.91)
Bio-Fast	ECL-LSDc	**0.82 (0.01)**	**10.28 (1.83)**	**818.90 (91.18)**
	ECL-LSDf	0.77 (0.01)	23.72 (2.19)	831.25 (57.03)
	ECL-GSD	**0.82 (0.03)**	10.66 (2.30)	1003.10 (37.47)
Bio-Slow	ECL-LSDc	0.68 (0.02)	**13.50 (2.57)**	916.00 (60.08)
	ECL-LSDf	**0.70 (0.02)**	25.40 (2.61)	**886.60 (43.72)**
	ECL-GSD	0.66 (0.01)	13.80 (2.50)	1034.00 (59.79)
Bio-Moderate	ECL-LSDc	**0.66 (0.01)**	**13.98 (3.57)**	935.20 (74.06)
	ECL-LSDf	0.62 (0.04)	25.02 (3.41)	**623.40 (57.60)**
	ECL-GSD	0.62 (0.05)	14.64 (2.13)	1002.60 (52.77)
Bio-Resistant	ECL-LSDc	**0.91 (0.01)**	**5.28 (1.21)**	**545.40 (46.71)**
	ECL-LSDf	0.90 (0.02)	12.56 (3.13)	588.50 (42.63)
	ECL-GSD	0.89 (0.01)	5.73 (2.87)	645.70 (27.12)

Table 7.22: Results of experiments on the relational datasets: average accuracy on the test sets, simplicity and training time in seconds (standard deviations between brackets).

fines the inequalities during the learning process using smaller intervals in order to take into account interdependencies between attributes.

7.7 Comparison with Other Systems

In this section we want to compare ECL with other systems for ICL both in the propositional and in the relational case. The results of ECL are those already presented in the previous sections of this chapter. For each dataset we have taken the best result obtained by the various settings of ECL. We used ECL-LSDc in all the cases but on the Mutagenesis dataset, where ECL-LSDf obtained better results. The EWUS selection operators was always used.

7.7.1 Propositional Datasets

We compare the performance of ECL on the propositional datasets with those of the following systems:

Dataset	ECL-LSDc	ECL-LSDf	ECL-GSD
Mutagenesis			
Accidents		•	(•)
Congestions		• ⋆	
Bio-Fast		• ⋆	
Bio-Slow			◦
Bio-Moderate		(•)	•
Bio-Resistant			•

Table 7.23: Results of t-test for the relational datasets. For each dataset we report when a method is significantly better than another, as estimated by t-test with confidence level of 1%, where • stands for ECL-LSDc, ◦ for ECL-LSDf and ⋆ stands for ECL-GSD. A symbol between brackets stands for a confidence level of 5%.

Method	ECL-LSDc	ECL-LSDf	ECL-GSD	Total
ECL-LSDc	-	3(4)	2(3)	5(7)
ECL-LSDf	0	-	1	1
ECL-GSD	0	2	-	2
Total	0	5(6)	3(4)	

Table 7.24: Results of the two-tailed paired t-test for the relational datasets with 1% confidence level: each entry contains the number of datasets on which the algorithm in the row is significantly better than the one in the column. The results of the test using 5% confidence level are reported between brackets when they differ from those using 1% confidence level.

C4.5 (Quinlan, 1993) is a landmark decision tree program. The decision tree is recursively grown starting from the root. The attribute with higher information gain is selected for becoming the root. If the attribute is nominal a branch is added for each value the attribute can assume. This splits up the example into subsets, one for every branch, using only those instances that actually reach the branch. If all the instances at a node have the same classification a leaf is added, with label equal to the classification of the examples, otherwise the process is repeated. If an attribute corresponding to a node is continuous then a two–way split is introduced, and the split point is the one with higher information gain;

IB1 (Aha et al., 1991) uses a simple distance measure to find the training instance closest to the given test instance, and predicts the same class as this training instance. If multiple instances have the same (smallest) distance to the test instance, the first one found is used. IB1 uses the Holte's discretization algorithm (see chapter 6);

HIDER* (Aguilar-Ruiz et al., 2003; Giráldez, 2004) is an EA that produces a hierarchical set of rules. When a new example is going to be classified, the set of rules is sequentially evaluated according to the hierarchy, so if the example does not fulfill a rule, the next one in the hierarchy order is evaluated. This process is repeated until the example matches every condition of a rule and then it is classified with the class that such rule establishes. An important feature of HIDER* is its encoding method (Giráldez et al., 2003): each attribute is encoded with only one gene, reducing considerably the length of the individuals, and therefore the search space size, making the algorithm faster while maintaining its prediction accuracy. HIDER* is also used in the first case study proposed in chapter 9;

GAssist (Bacardit and Garrel, 2003) is a Pittsburgh Genetic–Based Machine Learning system descendant of *GABIL* (DeJong and Spears, 1991). It evolves individuals that are ordered variable–length rule sets. The control of the bloat effect is performed by a combination of a rule deletion operator and hierarchical selection (Bacardit and Garrell, 2003). The knowledge representation for real–valued attributes is called Adaptive Discretization Intervals rule representation (*ADI*) (Bacardit and Garrel, 2003). This representation uses the semantics of the *GABIL* rules (Conjunctive Normal Form predicates), but using non–static intervals formed by joining several neighbor discretization intervals. These intervals can evolve through the learning process splitting or merging among them. The representation can also combine several discretizations at the same time, allowing the system to choose the correct discretizer for each attribute;

Naive Bayes (John and Langley, 1995) uses the Baye's rule of conditional probabilities combined with a "naive" presumption of conditional independence of attributes, to predict the class of examples;

SMO (Platt, 1999) implements the sequential minimal optimization algorithm for training support vector classifier. It only performs on binary classification. For classification, support vector machines (SVMs) operate by mapping the input into a feature space and by finding a hyperplane in the feature space. This hypersurface will attempt to split the positive examples from the negative examples. The split will be chosen to have the largest distance from the hypersurface to the nearest of the positive and negative examples. Intuitively, this makes the classification correct for testing data that is near, but not identical to the training data.

For C4.5, IB1, NaiveBayes and SMO we used the WEKA implementation (Witten and Frank, 2000b). For all the systems, we used the standard settings of WEKA in the experiments.

Table 7.25 reports the average accuracies obtained by the various systems on different propositional datasets, as estimated by ten–fold cross validation.

It can be seen that generally ECL obtained satisfactory results on the propositional datasets. ECL is never outperformed by C4.5, and on ten problems ECL

Dataset	ECL	C4.5	IB1	HIDER*	GAssist	NaiveBayes	SMO
Australian	**0.85 (1)**	**0.85 (4)**	0.81 (1)	**0.85 (3)**	**0.85 (5)**	0.77 (1)	**0.85 (1)**
Breast	**0.96 (2)**	0.94 (2)	0.95 (1)	0.96 (2)	**0.96 (2)**	**0.96 (1)**	**0.96 (1)**
Crx	0.85 (1)	0.85 (4)	0.81 (1)	0.83 (5)	**0.86 (5)**	0.76 (1)	0.85 (2)
Echocardiogram	0.78 (2)	0.71 (1)	0.67 (3)	**0.79 (13)**	0.72 (2)	0.75 (2)	0.75 (3)
German	0.74 (1)	0.72 (4)	0.67 (1)	0.73 (4)	0.72 (2)	0.75 (2)	**0.76 (1)**
Glass2	**0.85 (1)**	0.78 (4)	0.78 (1)	0.79 (3)	0.82 (8)	0.63 (1)	0.65 (2)
Heart	0.81 (2)	0.77 (4)	0.76 (2)	0.78 (8)	0.80 (7)	**0.83 (1)**	**0.83 (1)**
Hepatitis	0.83 (2)	0.79 (4)	0.81 (2)	0.83 (2)	**0.89 (8)**	0.83 (2)	0.85 (2)
Ionosphere	0.89 (2)	0.89 (7)	0.87 (1)	0.89 (6)	**0.93 (4)**	0.82 (2)	0.88 (2)
Liver	**0.67 (3)**	0.65 (1)	0.63 (1)	0.65 (4)	0.66 (8)	0.55 (1)	0.58 (1)
Pima-Indians	**0.77 (2)**	0.73 (3)	0.71 (1)	0.74 (2)	0.74 (2)	0.75 (1)	**0.77 (1)**
Sonar	0.76 (3)	0.73 (2)	**0.86 (1)**	0.73 (7)	0.75 (9)	0.68 (2)	0.78 (1)
Wdbc	**0.95 (3)**	0.94 (1)	**0.95 (1)**	0.94 (2)	0.94 (3)	0.93 (1)	0.94 (1)
Wpbc	**0.78 (4)**	0.72 (3)	0.71 (2)	0.76 (7)	0.75 (3)	0.67 (1)	0.76 (1)

Table 7.25: Average accuracies obtained by various systems for ICL on the propositional datasets. Standard deviation between brackets, where (x) stands for $(0.0x)$.

obtained better results. ECL generally outperforms IB1 as well. Only in two cases IB1 obtained better accuracies. In particular, on the Sonar dataset, IB1 obtained by far the best result among those obtained by all the other systems.

The performance of ECL and GAssist are comparable. GAssist outperformed ECL in four cases, namely on the Breast, Crx, Hepatitis and the Ionosphere datasets, while ECL obtained better results than GAssist in nine cases.

In eight cases, ECL obtains better results than HIDER*. In three cases (Glass2, Heart, Sonar) the difference is notable, while in the other cases the results are comparable.

Naive Bayes outperforms ECL only in two cases (German, Heart), while ECL obtains better results in eight cases, and in six of these cases (Australian, Crx, Glass2, Ionosphere, Liver and Wpbc) the results obtained by ECL are much better. On the opposite, in the cases where results obtained by Naive Bayes are better, they are comparable.

SMO obtains better, but comparable results in four cases and worse results in five cases. In three cases out of these five (Echocardiogram, Ionosphere and Wpbc) the results are comparable, while in the remaining two cases (Glass2 and Liver), the results achieved by ECL are of superior quality.

From the experiments it emerges that generally ECL and GAssist obtain the best performance, as far as accuracy is concerned, on the datasets used in these experiments.

Solutions found by GAssist are generally simpler than the solution found by ECL, while the simplicity of the solutions found by HIDER* is comparable to the simplicity of the solutions obtained by ECL, while solutions found by C4.5 are more complex than those found by ECL. It is difficult to compare the simplicity of the solutions found by IB1, SMO and NaiveBayes with the simplicity of the solutions found by ECL. This is because the systems do not

produce rules, and there is not a straightforward way to obtain rules from the models built by the systems.

As far as computational time is concerned, C4.5, IB1, GAssist, NaiveBayes and SMO are faster than ECL, while HIDER* requires a computational time comparable to the one required by ECL.

7.7.2 Relational Datasets

In this section we experimentally compare the performance of ECL with those of Progol, described in chapter 2, and with the performance obtained by the following two systems:

Tilde (Blockeel and De Raedt, 1997; Blockeel and Raedt, 1998) is an ILP system that induces hypotheses in the form of first–order logical decision trees. Tilde is an upgrade of C4.5 towards relational datamining. It builds decision trees that allow to predict the value of a certain attribute in a relation from other information in the database.

ICL (De Raedt and Van Laer, 1995) represents an upgrade of CN2 (Clark and Boswell, 1991) in order to learn first–order rules. CN2 is a propositional learner, that combines the advantages of the rule learner AQ and of the decision tree learner ID3 (Quinlan, 1986), i.e., it produces understandable rules and can cope with noisy data.

Both Tilde and ICL are not EAs and are part of the ACE-ilProlog datamining system (Blockeel et al., 2002).

Dataset	ECL	ICL	Tilde	Progol
Mutagenesis	**0.90 (0.01)**	0.88 (0.08)	0.86 (0.03)	0.88 (0.02)
Traffic	0.93 (0.02)	0.93 (0.04)	**0.94 (0.04)**	0.94 (0.03)
Biodegradability	**0.55 (0.03)**	**0.55 (0.02)**	0.52 (0.03)	0.53 (0.02)
Biodegradability2	0.74 (0.04)	**0.75 (0.01)**	0.74 (0.01)	0.71 (0.01)
Pyrimidines	**0.77 (0.02)**	**0.77 (0.03)**	0.76 (0.02)	0.75 (0.01)

Table 7.26: Average accuracy on relational datasets. Standard deviation between brackets.

In table 7.26 the average accuracies obtained by the various systems on the relational datasets are presented. In chapter 9 results for the Traffic dataset are discussed more extensively. Biodegradability2 is the Biodegradability dataset where the 4 classes are reduced to 2 classed (either degrade or resistant). The resulting dataset was first studied in (Van Laer, 2002).

ECL outperformed the other systems on the Mutagenesis dataset. On the same dataset Progol and ICL obtained the same accuracy of 0.88, while Tilde obtained the worst performance. On the Traffic datasets all the systems obtained similar results, with Tilde and Progol being able to obtain a slightly better accuracy.

All the systems obtained non satisfactory results on the Biodegradability dataset, when four classes are considered. On this problem the best results are obtained by ECL and ICL, but even these results are not satisfactory. When the four original classes are reduced to two, in the Biodegradability2 dataset, the performance of the systems improves evidently. In this case ICL obtained the best accuracy, which is slightly higher than the accuracy obtained by ECL. Progol, in this case, obtained the worst results. On the Pyrimidines dataset, ECL and ICL were able to obtain results of a slightly better accuracy. Also in this case the worst accuracy was obtained by Progol.

The simplicity of the solutions found by the four systems for the relational datasets is comparable, with Tilde being able of finding slightly simpler solutions.

Generally, all the systems obtained comparable results on all the datasets. In table 7.26 no results concerning the computational time are reported. We can however say that ICL and Tilde are much faster than ECL and Progol, with the first two systems requiring some seconds while the last two systems requires some minutes. ECL and Progol have comparable running times.

7.8 Conclusions

In this chapter we have experimentally evaluated various components of ECL. Moreover, we have compared the best results obtained by ECL in the various settings with the results obtained by other ICL systems, both in the propositional and in the relational setting.

A first set of experiments was aimed at evaluating the utility of using greedy mutation operators and the utility of incorporating an optimization phase. We have seen that including the optimization phase that follows the mutation phase and a degree of greediness in the mutation operators is beneficial for improving the accuracy of the found solutions. This is an important result, since it confirms the expectations behind this thesis: a system incorporating features of both standard GAs and of standard ICL systems can benefit from the complementary qualities of the two approaches. The drawback of this solution is represented by the computational time, that increases with the inclusion of greediness and of the optimization phase.

In section 7.3 we wanted to verify how using only a part of the available background knowledge at each iteration for reducing the computational cost of the learning process affects both the accuracy of the solution and the computational time. As expected, to lower values of pbk correspond lower computational times. What is interesting to notice, is that using more background knowledge does not necessarily imply obtaining solutions of higher quality. This is due to the capacity of EAs of dealing with noncontinuous search spaces. So even if little background knowledge on the problem domain is available, ECL can find solutions of reasonably good accuracy, and in some cases the best performance is obtained with low values of pbk.

The aim of section 7.4 was verifying the effectiveness of the variants of the

US selection operator for promoting diversity in the population. The EWUS selection operator obtains the best results in these terms. Also in terms of accuracy, the use of the EWUS selection operator generally leads to better results. This means that having more diversity in the population has a positive effect on the solution found by ECL.

Experiments for evaluating the three methods for extracting a final solution from the evolved population are presented in section 7.5. In chapter 5 we presented some examples illustrating the problems afflicting the procedures based on the heuristic for solving weighted set covering problems. From the results it emerges that these problems are present also when dealing with real-life ICL problems. The simple procedure based on precision is always capable of extracting the best solutions from the population. And the difference in the accuracy of the extracted solution is, in some cases, significant.

In section 7.6 results of experiments for evaluating the various discretization methods presented in chapter 6 are reported. From the experiments it emerges that ECL-LSDc is, on average, capable of finding solutions of better quality, both for propositional and relational datasets. The problem that afflicts ECL-LSDf is overfitting. In fact it is the best performing setting on the training sets, but the solutions found are too specific, and when applied to the test sets the quality is not as good as for the solutions found by ECL-LSDc. This is due to the fine grain initialization of inequalities in ECL-LSDf. In the future we intend to investigate other kind of initializations, for instance one that randomly chooses boundary points, in this way the overfitting phenomenon that afflicts ECL-LSDf may be resolved. The incapability of using class information on training examples negatively affects the performance of ECL-LUD. A possible variant of ECL-LUD could be designed, by changing the way in which the operators adopted for modifying inequalities act, in order to exploit information on the class of examples, turning in this way ECL-LUD into a supervised discretization method. This could be done, e.g., by estimating the density distribution of positive and negative examples inside each cluster and then use this information when modifying inequalities. An interesting result is that for the relational datasets the best performance was always obtained by either ECL-LSDc or by ECL-LSDf. This fact confirms what stated in chapter 6, i.e., that a global discretization method based only on entropy may not be suitable for some ILP problems.

Greediness	pbk	Selection	Solution Extraction	Numerical Values
ECL-Opt	domain dependent	EWUS	Precision	ECL-LSDc

Table 7.27: Summary of the results of the experiments aimed at assessing the components of ECL.

In table 7.27 we summarize the results of the experiments aimed at verifying the components of ECL. For each tested component, we report the best

setting of ECL.

In the last section of this chapter we have compared the results obtained by ECL with those of other systems, both for the propositional setting and for the relational setting. In both cases, ECL proved to be able to obtain performances that are comparable or better to those of other systems, both evolutionary and not evolutionary. The main drawback of ECL is the computational time required by the evaluation of individuals. In fact systems like C4.5 or GAssist for the propositional setting and Tilde and ICL for the relational setting are much faster that ECL. C4.5, Tilde and ICL are not EAs, and this is a reason why they are much faster than ECL. Regarding GAssist, which is a GA, the reason why it is faster than ECL is because ECL relies on Prolog for evaluating individuals, and the way this is actually implemented is not efficient and has to be improved.

On average, GAssist, C4.5, Tilde and ICL are about 20 order of magnitude faster than ECL. However, in the problems that we have tackled in this chapter, the most important aspect is the quality of the solutions found, and we have seen that in this term, ECL obtained good results, and in a resonable amount of time. Moreover, most machine learning tasks are offline. This means that speed is not a very relevant issue in these problems. A solution does not have to be provided in a small amount of time. We are interested mostly in the quality of the solution.

Chapter 8

Parallelization of ECL

In chapters 5 and 6 we have addressed the objectives 1–6, listed in figure 1.1 proposed in the introduction of this thesis. More specifically, we have addressed all the objectives regarding the effectiveness of the system, and one regarding the efficiency, i.e. the sampling of the background knowledge. In this chapter we address the last objective listed in figure 1.1, i.e. the parallelization of ECL. For more details about the implementation of the parallel version of ECL the reader can refer to (Staicu, 2003). The parallelization of ECL belongs to the objectives that regard the efficiency of the system, however we would also like to improve the effectiveness of ECL. Thus we can say that the parallelization of ECL has two aims:

1. reduce the computational cost of the learning process;

2. improve the accuracy of the found solutions.

With the parallelization of ECL, we expect to improve the accuracy of solutions because we intend to migrate individuals among different populations evolved in parallel. This should increment the diversity of the populations evolved, and, as we have seen in chapter 7, more diversity in the population generally corresponds to higher accuracy.

The first thing to be done is to decide how ECL can be parallelized. We have individuated two natural ways in which ECL can be parallelized. In a first way, the cost of performing several iterations of ECL is highly reduced, while in the second way the cost of evaluation when mutation and optimization are performed is reduced. In the following of this chapter we describe how this two parallelizations have been implemented. More precisely, in section 8.1, we begin by describing the island model, a popular method for parallelizing GAs. Section 8.2 describes the two parallelizations of ECL and the way in which processors can exchange individuals. Finally in section 8.3 an experimental evaluation is performed.

8.1 Island Model

There are various ways for parallelizing a GA, e.g., the farming or the island model (Whitley et al., 1997; Goldberg, 1989).

The farming model is centralized. A master process (the farmer) selects the best individuals from the population, and distributes them to a number of slave nodes (workers or slaves). Each slave applies genetic operators to the individuals it receives and then returns the evaluated individuals. In this way evaluation is parallelized, thus computational time is reduced.

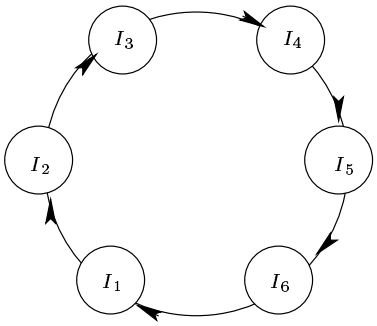


Figure 8.1: Example of ring topology for the island model.

In the island model, the population is divided into semi-isolated subpopulations, called demes. Each subpopulation is evolved in a node (an island), independently from other subpopulations. Islands can exchange individuals with a given frequency, e.g., at each generation. Usually the number of exchanged individuals is a small percentage of the size of the subpopulations. Typically the best individuals are migrated. When a node receives individuals, they are inserted into the receiving population. The most common replacement policy is to replace the worst k individuals in the receiving population with the k received individuals.

There are different topologies for implementing the island model, being the simplest a ring topology. In this topology every node has a right and a left neighbor. A node receives individuals from its left neighbor and sends individuals to its right neighbor. An example of ring topology is shown in figure 8.1. There are six nodes in this example, identified by I_i, $1 \leq i \leq 6$. The communication between nodes takes place in the direction indicated by the arrows. For example node I_1 sends individuals to node I_2 and receives individuals from node I_6. Another example of topology is to organize the nodes in a grid with wrap around. In this way each node has four neighbors, with which it exchanges individuals.

The communication among nodes can be synchronous or asynchronous. In the former case, when a processor has to migrate individuals, it sends some individuals of its population and waits for receiving individuals sent to them

by another processor. In this way all processors involved in the model are synchronized with the exchange steps. In the asynchronous case, each node is independent from the others. At each migration step, a node sends its individuals, but does not wait for receiving individuals. It simply checks if individuals have been received, and in this case they will be processed. If no individual has been received the processor continues with its tasks. If at the next exchange step individuals from the previous exchange step and from the current exchange step are received, only the newer individuals are considered. This is because it is assumed that they are better than their predecessors (they have evolved for a longer time). The second method is generally faster, since no time is spent waiting for receiving individuals. It is also fault tolerant, because even if a node is down, its neighbors can continue without having to wait for receiving individuals.

ALGORITHM(ECL)

```
1  ⋮
2  repeat
3        Evolve Population
4        Store Population into Final_Population
5    until max_iter is reached
6  ⋮
```

Figure 8.2: The first parallelization scheme acts on the main repeat statement of ECL.

8.2 Parallelizing ECL

We adopt an island model for a parallelization of ECL. A standard way of parallelizing a GA is to adopt an island model with a ring topology. The population is equally divided into a number of demes. Demes are then divided among the nodes that constitute the island model. So if the population size is pop_size and the number of nodes is m, each deme will have a size of $\frac{pop\_size}{m}$. If the sequential GA performs pop_size evaluations per generation, each node performs $\frac{pop\_size}{m}$ evaluations per generation. This because in standard GAs the entire population is evaluated at the end of each generation, thus dividing the population into subpopulations reduces the computational cost of evaluating individuals, which is the most time consuming operation performed by a GA.

For ECL dividing the population among the nodes will not reduce significantly the computational time, because at each generation only the individuals that are mutated are evaluated, and not the entire population. In figure 8.2 the main repeat statement of ECL is represented. In this cycle, a final population is

built as the union of *max_iter* populations. If we want to reduce the computational time of ECL, it is natural to parallelize the construction of these *max_iter* populations.

In the first parallelization of ECL, we divide the max_iter iterations of the main repeat statement among the nodes. Thus each node executes $\frac{max\_iter}{m}$ iterations. In the case that $m > max\_iter$, then max_iter is set to m and each node executes one iteration. It can be seen that the computational time is in this way reduced, especially in cases where max_iter is set to high values.

In our implementation, the island model is made of m nodes, where m is a user-tunable parameter, with default value equal to nine. We have chosen this default value because the parallel computer on which the parallel version of ECL is implemented, imposes a limit on the number of nodes that can be used by a single process, and because typical values of max_iter are lower than nine.

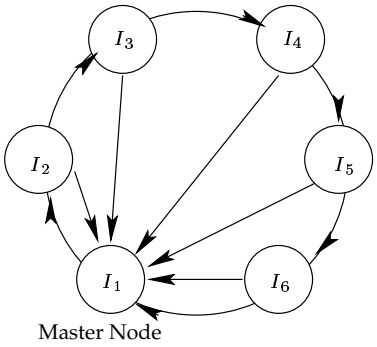


Figure 8.3: Final migration. All the nodes send individuals to the master node.

When a node has completed the iterations assigned to it, it will select and send a number of individuals from its evolved population to a master node. The master node performs a number of iterations, like the other nodes. The only difference is that it receives the individuals sent by all the other nodes when they have completed their iterations. The master stores the received individuals in a final population, together with the best individuals it has evolved. From this final population a solution is extracted. In order to extract the final solution, the master node applies the same procedure as the one applied in the sequential version of ECL, described in chapter 5. The situation is represented in figure 8.3, where the master node, in this case I_1, receives from the other five nodes a number of individuals.

The second way of parallelizing ECL we consider parallelizes the evaluations performed during mutation and optimization of individuals. Figure 8.4 shows how these operations are performed in the sequential version of ECL. n individuals are selected at each generation. Selected individuals are then mutated and optimized. These operations are performed sequentially and are time consuming. It is natural to parallelize these operations with a farming model.

ALGORITHM(ECL)

```
1  ⋮
2  for each selected chromosome φ
3  do Mutate φ
4     Optimize φ
5     Insert φ in Population
6  ⋮
```

Figure 8.4: For statement in which mutation and optimizations are performed.

Each node of the island model is assigned a number of slave processors. At every generation, the selected individuals are not mutated sequentially, but in parallel, by distributing them to the slaves. The farmer distributes the selected individuals among the slaves, and waits for receiving all the mutated and optimized individuals. The slaves perform mutation and optimization of the individuals they have received, and once these operations are performed, return the changed individuals to the farmer. In this way all the evaluations required by mutation and optimization are parallelized. Once received, the farmer inserts the individuals into the current population. In our implementation every node is assigned the same number of slaves. Typically each node is assigned four slaves. All the slaves use the same background knowledge assigned to their common farmer. The number of individuals assigned to each slave can be different. This happens when the number of individuals is not exactly dividable by the number of slaves. In this case some slaves will be assigned more individuals to mutate.

The original ECL implements a steady-states algorithm. This means that at each generation, individuals are selected, mutated and inserted in the population sequentially, one after the other. Thus there is the possibility that a already mutated individual is selected again in the same generation. In this second parallelization, if we want to reduce the computational time, this solution can not be adopted. Selected individuals have to be distributed among slaves, and for obtaining an improvement in computational time, all the mutations and optimizations have to be carried out in parallel. Thus in this parallelization, at each generation all the individuals are simultaneously selected and distributed to the slaves. Once received back, individuals are simultaneously inserted in the population. In this way an individual, once mutated cannot be selected again.

A scheme of the second parallelization is illustrated in figure 8.5. In the figure three slaves are assigned to each node. Slaves are denoted by s_i, $1 \leq i \leq 18$.

We perform experiments on two parallel versions of ECL:

PECL_1 implements only the parallelization of the main loop;

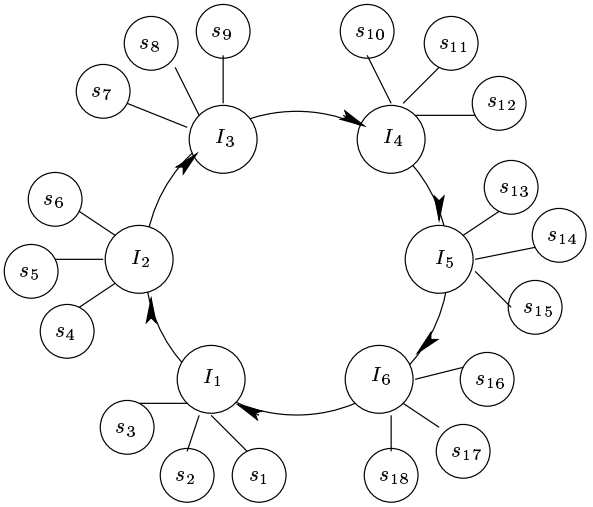


Figure 8.5: The second parallelization of ECL. Each node is assigned a number of slaves for performing mutation and optimization.

PECL_2 implements also the second parallelization we have just described.

8.2.1 Migrating Individuals

Individuals are exchanged between nodes at the end of each generation. Each node sends a number of individuals to its right neighbors and receives the same number of individuals from its left neighbors. This situation can be seen in figure 8.5, where, for instance, the node I_2 receives individuals from node I_1 and sends individuals to the node I_3. It is assured that the individuals sent by a node to its neighbor are all different from each other. Few parameters are needed for the migration of individuals.

Two parameters are used for setting the number of individuals that are migrated. Individuals with higher fitness and with higher precision are migrated at each generation. The number of individuals with higher fitness to migrate is supplied by the user by means of a parameter, m_f. Another parameter, m_p, is used for determining the number of individuals with higher precision that are migrated. Thus the total number of individuals migrated at each generation is $m_f + m_p$.

When a node receives individuals from its left neighbor, the received individuals are evaluated. This is necessary because typically the received individuals were evolved using a different portion of the background knowledge. This means that individuals that were good in one node may not be good in another node. Individuals are then inserted into the population, in the same way as new created individuals are, so by means of a tournament of size 4.

At the end of each generation each node performs the following operations:

1. select the best distinct individuals for migration;
2. move the selected individual to the right neighbor;
3. receive individuals from the left neighbor;
4. evaluate the received individuals;
5. insert the received individuals into the population.

Other two parameters, fm_f and fm_p, are used in the same way for controlling the final migration. So at the end of the process, the master node receives from each node $fm_f + fm_p$ distinct individuals.

Dataset	pop size	gen	sel	max_iter	Ni	lc	pbk	fm_f	fm_p
Australian	50	10	15	1	(4,4,4,4)	4	0.2	2	1
Breast	50	5	5	1	(3,3,3,3)	5	0.2	8	0
Glass2	150	15	20	3	(2,8,2,9)	5	0.2	0	2
Heart	50	10	15	1	(4,4,4,4)	6	1.0	3	0
Hepatitis	50	10	10	5	(4,4,4,4)	7	0.2	0	4
Ionosphere	50	10	15	6	(4,8,4,8)	6	0.2	2	1
Mutagenesis	50	10	15	2	(4,8,2,8)	3	0.2	4	0
Pima-Indians	60	10	7	5	(2,5,3,5)	4	0.2	2	1
Pyrimidines	50	15	10	10	(4,2,2,2)	5	0.2	2	1

Table 8.1: Parameter settings used in the experiments.

8.3 Experiments

Both PECL_1 and PECL_2 are implemented and tested on the DAS2 distributed supercomputer, where a network of high speed communication is used: Myrinet 2000 (Bhoedjang et al., 2003). The configuration of each node is the following:

- Two 1-Ghz Pentium-IIIs
- at least 1 GB RAM
- A Myrinet interface card
- A Fast Ethernet interface

All the experiments used a ring topology consisting of 9 nodes. In PECL_2, each of these 9 nodes is assigned 4 slaves, resulting in total of 45 processors. All the experiments were done using the ECL-LSDc method for treating numerical attributes (see chapter 6). Table 8.1 reports the other parameter settings used

in the experiments. In the following we report also results obtained by ECL. These results were obtained using the ECL-LSDc setting and the same parameter settings than the ones used for PECL_1 and PECL_2, and the runs were executed on the same machine.

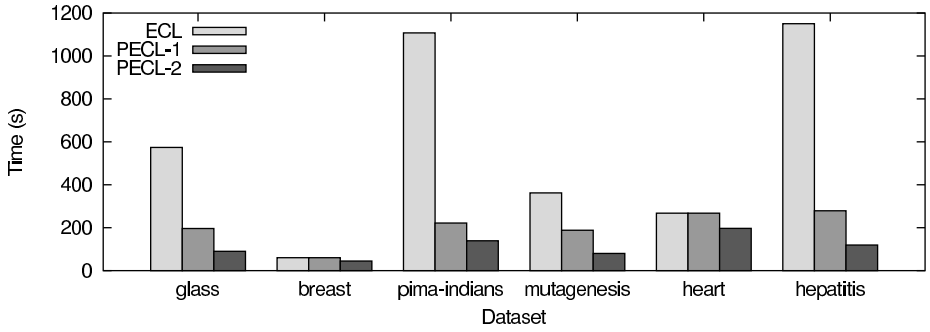


Figure 8.6: Times for the sequential and the two parallel versions of ECL.

The first aim of the parallelization of ECL was to reduce the computational cost of the learning process. Figures 8.6 and 8.7 reports the average computational time required by the three versions of ECL for performing one run. The three versions of ECL were run with the same parameter settings and on the same machine.

From the figures, it can be seen that the amount computational time required by the two parallel versions is much smaller than the one required by the sequential version of ECL, especially when the max_iter parameter is set to high values. When max_iter is set to 1, like on the breast and the heart dataset, the computational time required by ECL and PECL_1 is the same. The value of

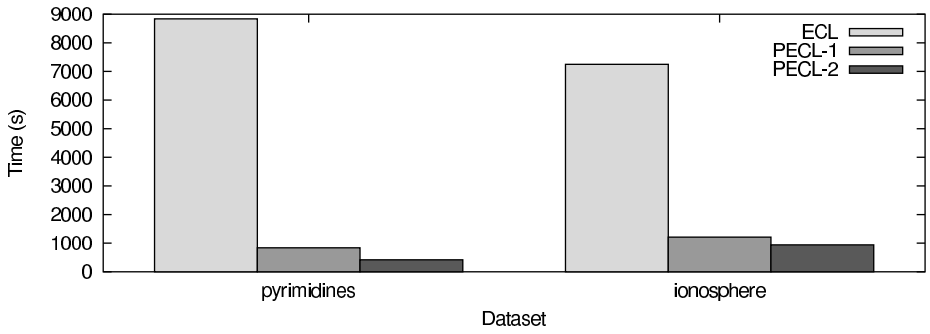


Figure 8.7: Times for the sequential and the two parallel versions of ECL for the Pyrimidines and the Ionosphere datasets.

max_iter for the breast and the heart datasets was set to 1, thus it follows that

PECL_1 can not be faster than the sequential version of ECL. This was expected, as we discussed in section 8.2. We have run these two experiments with this value for max_iter for verifying the impact of migration on the computational time. In fact the parallel version requires some extra time for migrating individuals. However the time required for this operation is insignificant compared to the time required for the evaluation of individuals. The time employed by PECL_2 is always inferior to the time required by the two other versions. This is explained by the fact that the time for executing mutation and optimization is highly decreased in PECL_2. So as far as computational time is concerned, PECL_2 obtained the best improvement.

Dataset	ECL	PECL_1	PECL_1m	PECL_2
Australian	**0.85 (0.01)**	0.85 (0.02)	0.85 (0.02)	**0.85 (0.01)**
Breast	0.95 (0.02)	0.95 (0.02)	0.95 (0.03)	**0.95 (0.01)**
Glass2	**0.85 (0.01)**	0.83 (0.01)	0.81 (0.02)	0.83 (0.02)
Heart	0.80 (0.03)	**0.82 (0.02)**	0.81 (0.02)	0.82 (0.02)
Hepatitis	**0.83 (0.02)**	0.83 (0.01)	0.81 (0.02)	0.81 (0.02)
Ionosphere	**0.89 (0.02)**	0.87 (0.02)	0.86 (0.02)	0.87 (0.02)
Mutagenesis	0.88 (0.01)	0.88 (0.01)	**0.89 (0.01)**	0.85 (0.03)
Pima-Indians	**0.76 (0.01)**	0.76 (0.01)	0.76 (0.02)	0.74 (0.01)
Pyrimidines	0.74 (0.01)	**0.75 (0.02)**	**0.75 (0.02)**	0.72 (0.01)

Table 8.2: Average accuracies obtained by the various version of ECL. Standard deviation between brackets.

The second objective stated at the beginning of this chapter, was to improve the accuracy of the found solutions. Table 8.2 shows the average accuracy obtained by the three versions of ECL on the datasets. The column labeled PECL_1m is relative to PECL_1 when migration between nodes is performed. The most precise individual in the population is migrated at each generation. We have conducted a number of experiments with different values of m_f and m_p and it emerged that the best results were obtained with $m_f = 0$ and $m_p = 1$.

We compare here the results obtained by PECL_1 and PECL_2 with the results of the sequential ECL.

A ten–fold cross validation is performed on each dataset, and three runs with different random seeds were performed on each fold. It can be seen that the performance of the sequential ECL and the parallel versions are comparable. It is interesting to notice that PECL_1m did not improve the performance of PECL_1. This is due to the fact that migrating individuals decreased the diversity in the populations evolved in the nodes instead of incrementing it. So as future work we plan to study different schemes of migrations, and different communication topologies in order to overcome these problems. A reason explaining the performance of PECL_2 is that in this variant, the algorithm implemented is not a steady-state algorithm anymore.

From the experiments performed it can be seen that the main objective of the parallelization has been achieved, since the parallel versions of ECL reduce the computational cost of the learning process. As far as the second objective is concerned, namely a better quality of the results, the parallelizations do not achieve the hoped results.

8.4 Conclusions

In this chapter two parallel implementations of ECL were described. An island model was used to this end, with individuals migrating among nodes. The aims of the parallelizations were to reduce the computational time of the learning process and to increase the accuracy of the found solutions.

As far as the first point is concerned, we can confirm that we have reached the objective. The computational time is reduced in both parallel versions of ECL. In particular in PECL_1 the computational time is reduced more and more as the number of iterations performed by ECL increases. In fact the first parallelization regarded only the execution of each iteration on a different node. With PECL_2 the computational time is always reduced, because in this case also each generation is parallelized, having the selected individuals mutated on a number of slaves assigned to each node.

Less successful was the achievement of the second objective, i.e., increasing the accuracy. From the presented results, it emerges that generally the accuracies of the solutions does not increase with the parallel versions. This can be due to the way in which individuals are sent to the master at the end of the evolution. Moreover, when individuals are migrated among the nodes, only the individuals are migrated, and no background knowledge is migrated. This can cause the following situation. Suppose that I_1 and I_2 are two nodes, and that φ is an individual being migrated form I_1 to I_2. In general the portion of background knowledge used in I_1 is different from the one used in I_2. This can mean that φ has a good fitness in I_1 but is considered rather poor in I_2. And this is because some part of the background knowledge needed for the evaluation of φ is not present in I_2. This can cause the elimination of φ from I_2. As already exposed, the reason for the poor performance of PECL_2 can be that in PECL_2 the algorithm is not steady–state anymore.

In future work, we intend to evaluate the possibility to migrate also background knowledge facts among the nodes, along with individuals. Also we want to evaluate different communication schemes among nodes. Another point to investigate is to promote co–evolution among the nodes, by means of a long term policy promoted by the master. For instance the master could control what part of the background knowledge to assigned to each node. The background knowledge assign to each node could be changed periodically, so that the focus of the genetic search performed on a node could be shifted toward other regions of the hypothesis space.

Chapter 9

Two Case Studies

In this chapter we present two case studies, a propositional and a FOL one. The first case regards the analysis of data about doctor–patient relationship and is a propositional case. In (Costa and Divina, 2003a) a preliminary study of this problem is described, while in (Aguilar-Ruiz et al., 2004) a more extended study is presented, and is here reported.

The motivations for carrying out this study is that the relationship between doctors and their patients is gaining more and more importance in the health care providing. It determines the compliance of the treatment and a part of the curative process. In the psychiatry the therapeutic relationship has even more power. Therefore having a general rule that could guide doctors towards a good relation with their patients would be very useful.

This first case study describes experiments in automated acquisition of such a rule by means of the application of ECL to the data. Moreover we apply three other ICL systems in order to extract knowledge from the data and validate the results of ECL.

In the second case study proposed in this chapter, we address the problem of automatic acquisition of knowledge about traffic problems. The task is to detect critical road sections by using information on road geometry and on data from sensor readings. A section is considered critical if an accident or a congestion took place on that section at a given time. Each section of roads have a number of sensors assigned to it. These sensors are used to measure a number of factors, e.g., the average speed, the average saturation and the average occupancy of the section. Moreover, information about the kind of road a section belongs to is available. For example, it is known if a section belongs to an highway, or to an in–ramp. All this information can be represented in a propositional way. However information about the geometry of the road requires a first–order logic representation. The geometry of the road network is given by information about what sections follow a given section. This is a many to many relation, since more sections can follow a single section.

For both problems, we report the best results obtained with the application of a system on all the available data, and then we perform other exper-

iments for validating the results as estimated by ten–fold cross validation, as it is done, e.g., in (Dolšak et al., 1994; Džeroski et al., 1998a; Džeroski et al., 1998b). We want to point out that in this kind of study, where the acquired knowledge is used by experts, the most important results are the best results obtained (Eiben and Smith, 2003a). Also the computational time needed for the acquisition of knowledge is not the most important factor in these kind of studies. For instance, in the second case study, we want to acquire knowledge for the automatic detection of critical sections. This knowledge will be then used in an expert system for traffic management. So the quality of the acquired knowledge is the most important factor, even if the computational time for the acquisition is high.

9.1 Analysis of Doctor–Patient Relationship

In the health care providing, satisfaction is earning more and more importance. Patients are nowadays better informed about their rights and the advances in medicine. They ask more from the doctor; the time when the patient believed everything that the doctor said is over. The traditional doctor–patient relationship is becoming a client–provider relationship. The way in which doctors approach patients becomes this way an important factor. In every branch of the medicine, the relation between doctor and patient is of great importance. It determines the compliance of the treatment and a part of the curative process. In the psychiatry the therapeutic relationship is even more critical. Therefore having a general rule that could guide doctors towards a good relation with their patient would be very useful. The communication between doctor and patient would be better and that would lead to more satisfaction, better compliance with the treatments and better prevention (Barker et al., 1996). That means an optimal use of health care services.

For these reasons we have decided to collect opinions from different patients with an interview questionnaire. The objective was to measure what the features of the doctor–patient relationship in different wards are and how different patients perceive it. In addition, the degree of satisfaction among the patients and the determinant factors for satisfaction were included in our objective. In our study we have taken two kinds of patients: psychiatric patients and non-psychiatric patients (belonging to the internal medicine and general surgery wards). The reason for these two groups is the special character of the doctor–patient relationship in the psychiatry. All the collected information has been organized in a dataset (Costa and Divina, 2003b).

In order to analyze the two problems, we use the following systems: C4.5, See5.0rules, HIDER* and ECL. A brief overview of the main features of C4.5 and HIDER* was given in chapter 7. See5.0rules (Quinlan, 2001) is the successor of C4.5, and can output *if–then* rules. It uses the same criterion to split the search space, although has been speeded up. The rules produced by See5.0rules are shorter than those obtained by the decision tree found with C4.5.

9.1.1 The Dataset

The data was collected between November 2000 and June 2001 in the psychiatry ward of the Hospital Gil Casares, Santiago de Compostela, Spain, and in the internal medicine and general surgery wards of the Hospital Xeral, Vigo, Spain. The same interviewer collected all data. Ninety patients were interviewed at the hospitals following an interview questionnaire.

The interview questionnaire was specially designed for this research project. The reason is that this questionnaire measures the satisfaction only from the doctor–patient relation point of view. Other questionnaires published in the medical literature, e.g., the Verona service satisfaction scale (VSSS) (Ruggeri and Dall'Agnola, 1993), take more elements into account. Our questionnaire is based on the VSSS, with less items to facilitate the recruitment of admitted patients and focusing on the doctor–patient relationship as a satisfaction determinant factor. The questionnaire consists on ten variables referred to patient's demographic data and twenty–six items that feature the relation between the doctor and the patient. Therefore, each patient is described by thirty–six attributes.

The ten demographic attributes include the sex, the age, the marital status (single, married...), the number of children, the place of residence, the education level, the working situation (if the patient is working or not, if the patient is retired...), the kind of profession and in which environment the patient lives (alone, with parental or own family).

The other twenty–six attributes include the satisfaction level of the relation between the interviewed patient and the doctor, the way the doctor gives information to the patient (the clearness of the language that the doctor uses with the patient), the frequency of the contact between doctor and patient, the degree of personal involvement of the doctor and so on. All this attributes are nominal.

A list of the attributes describing each patient, together with a brief explanation is given in table 9.1.

It was possible to answer to these twenty–six items with five different grades, according to the Likert's scale (Likert, 1932).

All this data, but the age of the patient, are discrete. The age of the patient is considered a continuous attribute. However when analyzing the dataset with ECL we have considered the attributes as continuous, and treated them with ECL-LSDf. This choice was made because in this way it is possible to have clauses that can assume the form of $case(X) : -var31(X, Y), 0.5 \leq Y \leq 2.5$, which is equivalent to say that the value for the attribute $var31$ can assume the values 1 and 2. In this way ECL have the possibility of letting an attribute assuming more values, as it happens, e.g., in HIDER*. For including new values in the description of an attribute ECL must include one value at a time, since the BP intervals are relative to just one attribute value. In the following we write that an attribute can assume a set of values as $\{v_1, \ldots, v_n\}$. For example the previous clause is written as $case(X) : -var31(X, \{1, 2\})$.

For privacy reasons each patient is identified by an unique code. So ex-

servicio: ward the patient belongs to
diagno: diagnosis made for the patient
sexo: sex of the patient
ec: marital status of the patient
edad: age of the patient
hijos: number of sons
SEPAR: if the patient is separated from the partner, where do the sons live
LUGAR: environment in which the patient live (city, village, village on the sea)
ESTUDIO: education level
LABORAL: working situation
PROFES: kind of profession
CONVIV: with whom the patient lives
VAR13: does the patient know the first and last name of his doctor
VAR14: sex of the doctor
VAR15: how many times the patient has been treated by his actual doctor
VAR16: how many time the patient has admitted in a hospital
VAR17: is the patient seen by the same doctor
VAR18: would the patient like to be seen by the same doctor
VAR19: has the doctor spoken with the patient's family
VAR20: would the patient like his doctor to speak with his family
VAR21: does the doctor speak of themes that are not related to the treatment
VAR22: would the patient like his doctor to speak of such themes
VAR24: would the patient like to change doctor
VAR25: does the doctor use words that are comprehensible to the patient
VAR26: would the patient like his doctor to speak in a clearer way
VAR27: does the doctor smile
VAR28: would the patient like his doctor to smile
VAR29: does the patient think his doctor understands the patient's problems
VAR30: would the patient like that his doctor would try to be in the patient's situation
VAR31: has the doctor explained to the patient in what the disease consists and its consequences
VAR32: would the patient like the doctor to explain him his disease
VAR33: has the doctor seen the patient outside office hours
VAR34: would the patient like to be seen outside office hours
VAR35: does the patient trust his doctor
VAR36: does the doctor let the patient wait
VAR37: if so how often does that happen
VAR38: would the patient like that the doctor apologizes when the patient is let to wait

Table 9.1: Description of the attributes of the doctor–patient relation dataset.

amples have the following form: $case(id_1), case(id_2) \ldots$, where id_1 and id_2 are different patients.

The background knowledge contains facts of the following form: *sex(*id_1*,m), age(*id_1*,30), satisfied(*id_1*,y), trust(*id_1*,1)* $\ldots$, saying that the patient identified by id_1 is a male of thirty years, satisfied with his relation with his doctor and that he considers that he can trust his doctor with a degree equal to 1, and so on.

9.1.2 Analysis of the Data

Each system is first applied to the entire dataset. The so produced rules were shown to domain experts and will be discussed in the following sections.

Table 9.2 shows the quality of the solutions found by the four systems. Pre-

	Satisfaction Problem			Service Problem		
System	Accuracy	Precision	Recall	Accuracy	Precision	Recall
ECL	0.90	0.88	**0.98**	0.89	0.84	0.95
C4.5	0.72	0.78	0.82	0.56	0.53	0.58
See5.0	0.89	0.89	0.95	0.83	0.82	0.84
HIDER*	**0.98**	**0.97**	0.97	**0.98**	**0.95**	**0.97**

Table 9.2: Accuracies, precisions and recall of the solutions found for the two problems.

cision, recall and accuracy are defined in chapter 5. Precision measures the proportion of examples that are correctly classified, while recall measures the proportion of positive examples that are correctly classified. Recall and precision are important, because these measures give information on how well a solution covered the positive examples (recall) and of how "noisy" this solution was (precision). Recall and precision are measures commonly used for evaluating information retrieval systems (Witten and Frank, 2000a).

In the following next to each rule information about the number of examples covered is given in the following form $[p_C/n_C]$. For example, in figure 9.1 the second rule correctly classifies 28 positive examples and makes only one mistake, so it incorrectly classifies as positive one negative example.

```
1:  satisfied(X)← var21(X,{0,1}),var35(X,{3,4}),ec(X,{3,4}).  [8/0]
2:  satisfied(X)← var24(X,0),var31(X,{3,4}).                  [28/1]
3:  satisfied(X)← var29(X,{3,4}),var18(X,{2,3,4}).            [44/4]
4:  satisfied(X)← var29(X,3).                                 [14/1]
5:  satisfied(X)← hijos(X,{2,3,4}),var24(X,{2,3}),var29(X,0). [2/1]
6:  satisfied(X)← var35(X,{3,4}),sexo(X,2).                   [23/4]
```

Figure 9.1: Results for the satisfaction problem obtained by ECL.

Satisfaction Problem

Patient satisfaction is related to the quality of the patient care. However, it is not the only parameter to consider in the measure of quality care (Weingarten et al., 1995). Satisfaction with health care is a multidimensional concept. It includes availability, financial aspects, convenience, staff's professional skills, among others. One of the determinant factors is the relation between doctor and patient (Mira, 2001).

In this problem positive examples are patients that are completely satisfied with their relation with their doctors, and negative examples are those patients who are not satisfied. For this problem there are 61 positive examples and 29 negative examples. The background knowledge contains 3274 facts.

Figure 9.1 shows the solution obtained by ECL for the satisfaction problem.

```
var29 = 0: n         [9/1]
var29 = 1
|   var33 = 0
|   |   var32 = 0: y [0/0]
|   |   var32 = 1: y [3/0]
|   |   var32 = 2: y [5/0]
|   |   var32 = 3: n [6/2]
|   |   var32 = 4: y [0/0]
|   var33 = 1: n     [4/0]
|   var33 = 2: y     [1/0]
|   var33 = 3: y     [0/0]
var29 = 2
|   var25 = 0: y     [3/0]
|   var25 = 1: n     [2/0]
|   var25 = 2: y     [0/0]
|   var25 = 3: y     [0/0]
|   var25 = 4: y     [0/0]
var29 = 3: y         [14/1]
var29 = 4
|   var18 = 0: y     [1/0]
|   var18 = 1: n     [3/0]
|   var18 = 2: y     [0/0]
|   var18 = 3: y     [2/1]
|   var18 = 4: y     [29/2]
```

Figure 9.2: Decision tree obtained by C4.5 for the satisfaction problem. Next to each leaf the number of examples that reaches the leaf and among those the number of example not belonging to the class predicted are shown.

If one clause is fulfilled then it means that the patient is satisfied. Some of the found clauses are interesting. For example, the second clause states that if a patient does not want to change doctor (var24) and if the doctor explains what the disease of the patient consists in (var31), then the patient is satisfied by the relation he has with the doctor. One can easily imagine that a patient wants to stay by the same doctor if he is satisfied with him. But it is not so evident how much the patient appreciates an explanation about his illness. We can assume that doctors in general always explain the diagnosis and its consequences to their patients. Nevertheless, the way patients perceive how this information is given to them can vary, and it can, according to this clause, have influence on the satisfaction of the doctor–patient relationship. The third clause is also interesting. It means that if the patient thinks that the doctor understands his problems (var29) and if the patient would like to be seen most of the times by the same doctor (var18), then the patient is satisfied.

We find here another important aspect in the relation between doctor and patient. A patient appreciates from a doctor not only his health advice, but also that he understands what the illness means for the patient in his personal circumstances.

The sixth clause states that female patients that trust their doctors (var35) are likely to be satisfied. This may suggest that having confidence in the doctor

is considered a more important factor for the satisfaction by women than by men.

```
Rule 1:  var18 = 4
         var29 = 4  → class y  [29/2]
Rule 2:  var29 = 3  → class y  [14/1]
Rule 3:  var33 = 2  → class y  [6/0]
Rule 4:  var29 = 1
         var32 = 2  → class y  [6/0]
Rule 5:  var29 = 1
         var33 = 1  → class n  [4/0]
Rule 6:  var29 = 0  → class n  [9/1]
Rule 7:  var18 = 1
         var29 = 4  → class n  [3/0]
Rule 8:  var29 = 1
         var32 = 3  → class n  [7/2]
Default class:  y
```

Figure 9.3: Decision rules obtained by See5.0rules for the satisfaction problem.

```
R1:
        diagnoIS NOT 10
        ec    IS NOT 4
        var16 IS NOT 2
        var17 IS NOT 1
        var19 IS NOT 4
        var29 IS NOT 3
        var35 IS NOT 3 → class n  [18/1]
ELSE R2:
        var17 IS NOT 0
        var18 IS NOT 1
        var24 IS NOT 3
        var26 IS NOT 2
        var35 IS NOT 2 → class y  [52/1]
ELSE R3:
        diagnoIS NOT 6
        hijos IS NOT 4
        var13 IS NOT 2
        var20 IS NOT 1 → class y  [7/0]
ELSE → class n [10/1]
```

Figure 9.4: Decision rules obtained by HIDER* for the satisfaction problem.

Figure 9.2 shows the decision tree obtained by C4.5 for the satisfaction problem. From the depicted tree, a decision rule can be obtained following each branch of the tree. For example, if we follow the path individuated by the branches `[var29=1,var33=0,var32=2]` we reach a leaf labeled with a `y`, which stands for *satisfied*. We obtain in this way a decision rule with three conditions (on the attributes `var29, var33` and `var32`) that if fulfilled by an example classifies the example as a satisfied patient. This rule correctly clas-

sifies 5 positive examples and makes no mistakes. This information can be found next to each leaf of the tree. In the same way we can follow the other branches and obtain a set of decision rules. Notice that some leaves cover no examples at all (such leaves have a [0/0] next to them). These branches are present because C4.5 adds a branch for every value the attribute that individuates a subtree can assume. An interesting rule that can be derived from the path `[var29=4,var18=4]`, that states that if a patient thinks his doctor understands his problems and if the patient wants to be seen by the same doctor then he is satisfied. We can derive from the tree a set of rule that is similar to the third rule found by ECL. We can also derive the fourth clause of figure 9.1 from the tree obtained by C4.5.

Figure 9.3 represents the rules obtained by See5.0rules. In these rules the conditions on the attributes are on the left of the `->`. For example the first rule is exactly the same rule that the one we have derived from the decision three obtained by C4.5 following the path `[var29=4,var18=4]`. These rules also confirms the fact that the belief of the patient that his doctor understands his problems (var29) is an important factor for the satisfaction of the patient. Another aspect that seems to be important for a good relation doctor–patient is the frequency with which the patient is seen by the same doctor (var18). Patients that are seen most of the times by the same doctor tends to be more satisfied. This could be because satisfied patients go always to the same doctor, or because the continuity in the care leads to satisfaction of the patient.

The output of HIDER* is shown in Figure 9.4. Decision rules provided by HIDER* presents an original feature: they are hierarchical, so the application of each one must be in specific order. The confidence inspired by the doctor (var35) and whether the patient visits often the same doctor (var17) seem to have relevance in this rule set. In addition, the fact that the same doctor treats always the patient and that the patient would not like to change to another doctor have influence on the classification.

Service Problem

In this problem we want to induce rules to distinguish psychiatric patients from non–psychiatric patients. Psychiatric patients differ from other kind of patients in demographic features. Disorders as schizophrenia, alcohol and drugs abuse are more frequent in the population with a lower social status. Bipolar disorder type I (DSM-IV, 2000) is more frequent among people with lower educational level. Dementia is more frequent among the elderly (Kaplan et al., 1994).

Perhaps these differences go further than the demographic data. Thus it would be interesting to know if psychiatric patients perceive the doctor–patient relation differently than others.

For this problem there are 43 positive examples (psychiatric patients) and 47 negative examples (non–psychiatric patients). The background knowledge is made of 3198 facts. The class of each example is determined by the unit to which a patient belong. There are less facts in the background knowledge

because some attributes were excluded from this problem, e.g., the diagnosis for the disease of the patient was excluded, for obvious reasons. The quality of the solutions found are shown in table 9.2.

```
1:  case(X)← var18(X,{0,1}), var31(X,{0,1}).            [9/0]
2:  case(X)←var19(X,4),var27(X,{0,1,2,3}).              [8/0]
3:  case(X)←edad(X,Y),var25(X,{0,1}),(32.5 <Y ≤ 36.5). [4/0]
4:  case(X)←var15(X,{1,2}),var33(X,0).                  [10/1]
5:  case(X)←laboral(X,{0,1}),var30(X,3).                [11/2]
6:  case(X)←var31(X,0),var32(X,{0,1,2}).                [17/2]
7:  case(X)←laboral(X,{2,3}).                           [12/4]
8:  case(X)←var14(X,2),var15(X,{0,1,2}).                [8/2]
9:  case(X)←var17(X,4),satisfecho(X,{0,1}).             [6/2]
```

Figure 9.5: Results for the service problem obtained by ECL.

The solution induced by ECL for the service problem is shown in figure 9.5. The first and the second clauses are interesting. The first clause states that if a patient does not like to be visited by the same doctor (var18) and if the patient affirms that he has not be given explanations about his illness (var31) then he is likely to be a psychiatric patient. This rule will be discussed in the following. The second clause considers also if the doctor smiles. However the values this attribute can assume are almost all the possible ones, so we can consider this feature as not important for this clause. Thus we can say that this clause states that if the doctor has contacted the member of the patient's family (var19), than the patient is a psychiatric patient. This can be explained by the fact that psychiatrists often need extra information from the patient's environment in order to make a diagnosis: behavioral changes are usually better perceived by the family than by the patient himself. Information about patient's development as a child can sometimes be provided only by his mother. If the patient is confused, his family could explain what has been really happening to the patient in the last time. On the other hand, in other medical specialties, environmental information is not so crucial for the diagnosis.

The decision tree depicted in figure 9.6 represents the solution found by C4.5, while the solution obtained by See5.0rules is given by the rules shown in figure 9.7. In both C4.5 and See5.0rules cases the class predicted by the rules can be either p for psychiatric patients or n for non psychiatric patients. It can be seen that the two systems found a rule that states that if a patient is not given any explanations about his disease (var31) then he is likely to be a psychiatric patient. This rule is found in the first branch of the tree shown in figure 9.6 and in the second rule of figure 9.7. It correctly classifies 26 psychiatric patients but makes 7 mistakes. The first clause found by ECL is similar to this one.

From these rules it emerges that explanations given by the doctor to the patient regarding his disease is an important factor for distinguishing between psychiatric and non psychiatric patients. If a patient is not given any explanation then he is likely to be a psychiatric patient. This rule could be explained by the fact that psychiatric patients are sometimes confused, or have little insight

```
var31 = 0: p         [26/7]
var31 = 1
|   var25 = 0: n     [9/1]
|   var25 = 1: p     [2/0]
|   var25 = 2: n     [0/0]
|   var25 = 3: n     [0/0]
|   var25 = 4: n     [1/0]
var31 = 2: n         [8/1]
var31 = 3
|   ec = 0: n        [0/0]
|   ec = 1: p        [2/0]
|   ec = 2: n        [3/0]
|   ec = 3: n        [0/0]
|   ec = 4: n        [0/0]
|   ec = 5: n        [0/0]
var31 = 4
|   var27 = 0: n     [0/0]
|   var27 = 1
|   |   var38 = 0: p [1/0]
|   |   var38 = 1: n [3/0]
|   |   var38 = 2: p [0/0]
|   |   var38 = 3: p [0/0]
|   |   var38 = 4: p [4/0]
|   var27 = 2
|   |   var19 = 0: p [0/0]
|   |   var19 = 1: p [2/0]
|   |   var19 = 2: n [2/0]
|   |   var19 = 3: p [0/0]
|   |   var19 = 4: p [0/0]
|   var27 = 3: p     [2/1]
|   var27 = 4: n     [12/0]
```

Figure 9.6: Decision tree obtained by C4.5 for the service problem.

(realistic conception of their illness). That could mean that even if the doctor has given the patient explanations about their diagnosis, the patient does not perceive it as a valid one and thinks that nobody has informed him.

Figure 9.8 shows the results of HIDER* for the service problem. The level of education (estudio) is relevant to know whether an example is positive or not. When the patient has university degree (rule 1) or high school studies (rule 2) is more likely to not be a psychiatric patient. The number of times the patient has been treated by the doctor (var15) is important in rule 2, as well as the will of the patient to have his doctor to talk to his family. Interestingly, the doctor does not apologize when the patient is not a psychiatric one, which suggests that doctors are more concerned about their relation towards patients in a psychiatric ward.

Validation

In order to validate the results obtained by the four systems, a ten–fold cross validation is used. Ten runs with different random seeds are performed on

```
Rule 1:  var27 = 1
         var38 = 4 → class p  [8/0]
Rule 2:  var31 = 0 → class p  [26/7]
Rule 3:  var27 = 1 → class p  [22/10]
Rule 4:  ec = 1    → class p  [22/12]
Rule 5:  var31 = 1 → class n  [10/3]
Rule 6:  ec = 2    → class n  [30/13]
Rule 7:  var31 = 4 → class n  [19/10]
Default class:  n
```

Figure 9.7: Decision rules obtained by See5.0rules for the service problem.

```
R1:
        estudio IS NOT 4
        laboral IS NOT 4
        var24   IS NOT 1
        var27   IS 0,1,3
        var29   IS NOT 3
        var37   IS NOT 3 → class p  [29/1]
ELSE R2:
        estudio IS NOT 3
        laboral IS NOT 2
        conviv  IS 0,1,2
        var15   IS NOT 0
        var20   IS NOT 2
        var37   IS NOT 4 → class p  [12/0]
ELSE → class n [46/2]
```

Figure 9.8: Decision rules obtained by HIDER* for the service problem.

each fold.

Table 9.3 reports the results obtained by the four systems on unseen cases, as estimated by ten–fold cross validation. Standard deviations are shown between brackets. The column label "Rules" reports the average number of rules forming the solution found.

C4.5 obtained the worst performance on both problems. Especially for the service problem the performance of C4.5 are much worse than the ones obtained by the other systems. ECL and HIDER* obtained the best accuracy for the satisfaction problem, and for the service problem the accuracy of ECL and HIDER* are comparable to the one obtained by See5.0.

The computational cost is the main drawback of ECL and HIDER*, as they are evolutionary–based techniques, being the C4.5 and See5.0 much faster. However, in this sort of problems, the computational cost is not a determinant aspect, the quality of the solution found is the most important factor. Regarding the simplicity of the solutions found, ECL and HIDER* obtained very simple solutions.

	Satisfaction Problem			Service Problem		
System	Accuracy	Time	Simplicity	Accuracy	Time	Simplicity
ECL	**0.78 (0.04)**	357.1	4.3 (0.67)	0.71 (0.04)	388.91	6.2 (0.98)
C4.5	0.72 (0.05)	0.26	18.12 (1.73)	0.55 (0.03)	0.10	24.93 (1.34)
See5.0	0.76 (0.03)	0.06	6.33 (0.26)	0.73 (0.02)	0.06	4.29 (0.03)
HIDER*	0.77 (0.13)	1734	4.2 (0.4)	**0.76 (0.11)**	1221	4.0 (0.6)

Table 9.3: Results for ten–fold cross validation. Time is expressed in seconds. In the column labeled simplicity the average number of rules forming the solution is given.

9.1.3 Conclusion for the First Case

In this first case study, we have analyzed a new real life dataset regarding the relation doctor–patient. We have derived two problems from this dataset: the satisfaction of the patient from his relation with the doctor and the problem of identifying psychiatric patients. This last problem is important because analyzing the induced rules, experts can check what the differences are in the doctor–patient relationship with psychiatric patients.

For the satisfaction problem the systems performed at roughly the same level as far as accuracy is concerned. From the obtained results we can conclude that determining factors for satisfaction with the doctor–patient relationship are the feeling of being understood by the doctor and hearing from the doctor what the diagnosis and its implications are. It seems that other aspects, such as demographic features of the patients, waiting time and the language the doctor uses with the patients, do not determine the satisfaction level.

For the service problem, the found rules are not easily explainable. This could mean that psychiatric patients are not so different from other kind of patients in their expectations towards the therapeutic relationship. According to one system, only the level of studies seems to be a differentiating aspect between these types of patients.

In this study it was important to apply more than one technique in order to analyze the data. This is because in this way we have the possibility to check if the rules found by a system are in some way complementary to the rules found by the other systems. In our case we have seen that the systems estimated as important the same set of attributes for classification in both the problems addressed. The two evolutionary techniques (ECL and HIDER*) used in this study found better solutions than the others, based on entropy measures.

Knowledge models, as those presented in this section, provide important insight to medical researchers in order to better understand psychiatric aspects from patients. In general, these techniques are useful to predict the type of patient and to understand which features become relevant for psychiatric patients, which could be analyzed in detail by doctors to improve the health care.

9.2 Detecting Traffic Problems

The second case study we address was first proposed in (Džeroski et al., 1998a; Džeroski et al., 1998b). This problem has an application in expert systems for decision support in road transport management. The goal of such systems is to advise traffic management center operators by proposing control actions to eliminate or reduce traffic problems according to the global state of traffic. In order to assess the global state of traffic, the system periodically receives readings from sensors that are distributed on the road network. The sensors measure magnitudes such as speed, flow (the number of vehicles on that section per hour) and occupancy (percentage of time that the sensor is occupied by vehicles), and other information about the current state of control devices, such as traffic lights. The system then interprets the received signals, detects a possible traffic problem, gives the possible cause and proposes recommendations on how to solve or reduce the problem.

The usual approach to building traffic expert systems is to use knowledge based architectures that support the strategies of reasoning followed by operators. This approach requires, among other things, the use of knowledge for detecting traffic problems. Knowledge about different traffic scenarios, e.g., accidents or congestions, can be used to generate or improve the knowledge base for problem detection of the expert system.

In this case study we address the problem of acquiring such knowledge for detecting traffic problems. In particular the problems that we consider are accidents and congestions.

In the following we first describe the dataset, and then we compare the rules extracted by ECL with the rules extracted by Progol and ICL. This dataset has been used in chapter 7 for both assessing the effectiveness of the various discretization methods that can be used by ECL and for assessing the global effectiveness of ECL.

Progol was described in chapter 2, and ICL was briefly introduced in chapter 7. Both Progol and ICL can produce rules that are easy to interpret, as are the rules induced by ECL. On the opposite, the trees induced by Tilde are rather difficult to interpret, and so they will not be presented in this study.

9.2.1 The Dataset

The dataset subject of this study was generated with the AIMSUN (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks) system (Ferrer and Barcelo, 1993), a software able to reproduce the real traffic conditions of any urban network on a computer. A model of the urban-ring of Barcelona was developed using this simulator. The model includes the same variables that the real information system TRYS (Cuena et al., 1995) records using sensors and was calibrated using information from the real system. TRYS is an expert system for traffic management developed for the cities of Madrid and Barcelona.

The road network is represented in an object oriented way, where sections are the basic objects. A number of sensors are associated to each section. Several kind of sections are defined, e.g., off–ramp, on–ramp or highway. The geometry of the road network is described by relations, that describe when a section follows or precedes another section. The information about the sections and the kind of section is a static information. The sensors associated to each section provide a continuous stream of information, sending five readings per minute. Information about the flow (defined as the number of cars that passed the sensor in the last minute), the occupancy (defined as the proportion of time the sensor is occupied, in thousandths) and the average speed of the cars that passed the sensor during the last minute are provided. Saturation is a derived quantity defined as the ratio between the flow and the capacity of the section. The capacity of a section depends on the number of lanes of the section.

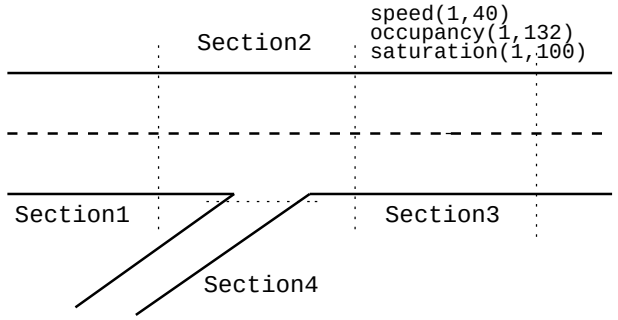


Figure 9.9: An example of highway divided in three sections and with another section that represent an on–ramp. A number of sensors is associated to each section. These sensors provide readings regarding different magnitudes, as shown for the third section.

A graphical example of how information describing the road network is represented is shown in figure 9.9. In the figure a highway and an on–ramp (`Section4`) are represented. Four sections are shown in the picture, and are identified by vertical dashed lines. For each section sensors provide readings regarding different magnitudes, such as speed, occupancy and saturation. For instance, we know that, at minute one on section `Section3`, the average speed was 40 Km/h, the occupancy rate was 132 and the average saturation was 100.

The created dataset consists of 66 examples of congestions, 62 examples of accidents and 128 examples of sections where no problem took place. The last sections are called non-critical sections. Each example describes the situation at a given time on a given section. For instance `accident(Section1,Time)` means that there was an accident on `Section1` at time `Time`. In a similar way `congestion(Section2,Time)` states that there was a congestion on `Section2` at time `Time`.

Each section has a type associated to it, which can be highway, off-ramp,

on-ramp. This information is represented by a predicate of the following form:

```
tipo(Section,Type).
```

where `Type` can assume the following values: `carretera` (highway), `rampa_incorporacion` (on–ramp) and `rampa_abandono` (off–ramp). The relational structure of the road network is described by the following predicate:

```
secciones_posteriores(Section1,Section2).
```

that states that `Section2` follows `Section1`, as it happens, e.g., in figure 9.9. This is a many to many relation, as more than one section can follow another section. For instance, in figure 9.9 `Section2` and `Section4` follow `Section1`. All this information forms part of the background knowledge.

The background knowledge contains also facts relative to sensor readings, which are in the form:

- `velocidad(Time,Section,Value)` this predicate describes the average speed, expressed by `Value`, on section `Section` at time `Time`;
- `ocupacion(Time,Section,Value)` this predicate describes the the average occupancy, expressed by `Value`, of section `Section` at a given time `Time`;
- `saturacion(Time,Section,Value)` this predicate describes the average saturation, expressed by `Value`, of section `Section` at time `Time`.

Reading relative to 144 minutes were produced, thus `Time`$\in [0, 143]$.

	Speed	Occupancy	Saturation
low	$Value \leq 45$	$Value \leq 125$	$Value \leq 42.5$
medium	$45 < Value \leq 75$	$125 < Value \leq 275$	$42.4 < Value \leq 77.5$
high	$Value > 75$	$Value > 275$	$Value > 77.5$

Table 9.4: Values for discretization used in the original study.

In the original study the argument `Value` of the three predicates was discretized in three values: high, medium and low. The ranges of values used for the discretization are given in table 9.4, for example speed was considered to be low if the value was lower than 45 Km/h.

The use of ILP for this dataset is justified by the presence of information about road geometry. A propositional learner can not exploit this information.

9.2.2 Analysis of the Data

ECL was first applied to the entire dataset in order to induce rules for identifying traffic problems. Then the system is validated using ten–fold cross validation, and its performance are compared to those obtained by other systems.

```
accident(X,T)← saturacion(T,X,S_X),velocidad(T,X,V_X),
               S_X ≤ 43.5,V_X ≤ 63.        [31,0,0]
accident(X,T)← velocidad(T,Y,V_Y),tipo(X,carretera),
               saturacion(T,X,S_X),secciones_posteriores(Y,X),
               V_Y ≤ 40.75,S_X ≤ 49.125.   [28,0,0]
accident(X,T)← saturacion(T,X,S_X),ocupacion(T,X,O_X),
               S_X ≤ 49.125,O_X ≤ 984.25.  [16,0,3]
accident(X,T)← ocupacion(T,X,O_X),saturacion(T,X,S_X),
               S_X ≤ 46.1875,O_X > 818.875. [36,0,3]
```

Figure 9.10: The logic program obtained by ECL for describing accidents.

In this section we present and compare the rules obtained by ECL, Progol and ICL on the entire dataset.

In this case study we use ECL-LSDc (see chapter 6), so where the discretization is not a global discretization, as in (Džeroski et al., 1998a), but is the result of a local supervised discretization described in chapter 6. Only for the speed values utilized when learning rules for the congestion class the discretized values given by experts were used. This is because it was not possible to determine the BP and DP intervals in this case, since speed reading are not associated to sections where a congestions occurred. In fact if there is a congestion on a section the speed is zero or very low. In all the other cases BP and DP intervals were utilized in the local supervised discretization.

```
accident(X,T) ← velocidad(T,X,baja),
                saturacion(T,X,baja).  [27,0,0]
accident(X,T) ← secciones_posteriores(Y,X),tipo(X,carretera),
                velocidad(T,Y,baja).   [29,0,0]
accident(X,T) ← ocupacion(T,X,alta),saturacion(T,X,baja),
                secciones_posteriores(X,Y),
                velocidad(T,Y,alta).   [36,0,0]
```

Figure 9.11: Rules for the accident class induced by Progol.

As explained in chapter 5, the input of ECL consists of a set of positive and negative examples and a background knowledge. In this problem there are three classes of examples: `accident`, `congestion` and `noncs` (for non–critical section). A set of rules is induced for each class. Positive examples of one class are treated as negative examples of the other classes. For instance when learning a theory for the class `congestion`, the positive examples for `accident` and for `noncs` are used as negative examples. Results are evaluated as explained in chapter 5.

Figure 9.10 shows the clauses induced by ECL for describing sections where an accident took place. The first rule states that there is an accident on a section X at time T if the saturation reading is less than 43.5 and if the average speed on the section is less than 63 Km/h. The second rule states that if the saturation level at a given time T on a section X of highway is less than 49.125 and on a

```
class(accident) ← section(X),timemoment(T),tipo(X,carretera),
                  secciones_posteriores(Y,X),
                  velocidad(T,Y,baja).          [29,0,0]
class(accident) ← section(X),timemoment(T),ocupacion(T,X,alta),
                  saturacion(T,X,baja),velocidad(T,Y,alta),
                  secciones_posteriores(X,Y).   [36,0,0]
class(accident) ← section(X),timemoment(T),velocidad(T,X,baja),
                  secciones_posteriores(Y,X),
                  ocupacion(T,Y,alta).          [27,0,0]
class(accident) ← section(X),timemoment(T),ocupacion(T,X,alta),
                  saturacion(T,X,baja),secciones_posteriores(X,Y),
                  tipo(Y,carretera),secciones_posteriores(Y,Z),
                  ocupacion(T,Z,baja).          [22,0,0]
```

Figure 9.12: Rules generated by ICL for the accident class.

previous section Y the reading of the speed at time T is less than 40.75 then there is an accident on section X.

The last two rules cover also examples of the non–critical class. However 2 of the negative examples covered by the last clause are covered also by the third rule, so the two clauses cover four examples that belong to the non–critical class.

```
congestion(X,T)← velocidad(T,Y,V_Y),secciones_posteriores(Y,X),
                 tipo(X,rampa_abandono),ocupacion(T,X,O_X),
                 V_Y ≤ 45),64.125 < O_X ≤ 628.75.   [0,30,0]
congestion(X,T)← ocupacion(T,X,O_X),tipo(X,rampa_incorporacion),
                 saturacion(T,X,S_X),410.75 < O_X ≤ 779.25,
                 56.5 < S_X ≤ 82.625.               [0,27,0]
congestion(X,T)← saturacion(T,Y,S_Y),secciones_posteriores(X,Y),
                 ocupacion(T,X,O_X),tipo(X,rampa_incorporacion),
                 S_Y > 77.25,260 < O_X ≤ 779.25.    [0,32,4]
```

Figure 9.13: Rules obtained by ECL for congestion.

Figure 9.11 shows the rules induced by Progol. It can be seen that the first rule imposes conditions on the same measures as the first rule induced by ECL. The conditions are different however, because the rule found by Progol require an average speed lower than 45 Km/h and a saturation lower than 42.5. In this case, the rule obtained by ECL without the discretization covers more cases of accidents. The second rule recalls the second rules induced by ECL. The rule induced by Progol imposes less conditions namely it does not impose any condition on the measurement of the saturation for section X. Besides, the condition on speed imposed by ECL requires an average speed value lower than the one imposed by Progol.

Figure 9.12 shows the rules induced by ICL. The first and the second rules obtained by ICL are equal to the second and the third rules found by Progol, respectively. The rules obtained by ICL and Progol are more precise than the

```
congestion(X,T)←tipo(X,rampa_abandono),velocidad(T,Y,baja),
                secciones_posteriores(Y,X). [0,30,0]
congestion(X,T)←secciones_posteriores(X,Y),
                saturacion(T,Y,alta),secciones_posteriores(Z,Y),
                velocidad(T,Z,baja).       [0,31,0]
```

Figure 9.14: Rules induced by Progol for the congestion class.

ones obtained by ECL.

Figure 9.13 shows the rules obtained by ECL for detecting congestions on sections of road. The first rule states that there is a congestion on a section X at time T, if X is a off–ramp, the occupancy rate on X at time T is between 64.125 and 628.75 and if on a previous section Y the speed was less that 45 Km/h, so it was low. The range for the occupancy level includes almost all the possible values, so in this rule the most important factor is the average speed on the previous section. The second rule states that there is a congestion on section X if the section is an on–ramp and the saturacion and the occupancy rates are those shown in the rule.

```
class(congestion) ← section(X),timemoment(T),
                    tipo(X,rampa_incorporacion),
                    secciones_posteriores(X,Y),
                    saturacion(T,Y,alta),
                    secciones_posteriores(Z,Y),
                    velocidad(T,Z,baja).  [0,31,0]
class(congestion) ← section (X),timemoment(T),
                    tipo(X,rampa_abandono),
                    secciones_posteriores(Y,X),
                    velocidad(T,Y,baja).  [0,30,0]
```

Figure 9.15: Rules for the congestion class obtained by ICL.

The rules obtained by Progol for describing the congestion class are shown in figure 9.14. It can be seen that the first rule is almost the same as the first rule obtained by ECL. The only difference is that the rule obtained by Progol does not impose any condition on the occupancy rate of the section being described. However, as already stated, the accepted range of values of the rule obtained by ECL include almost all the possible values, thus the two rules can be considered as equivalent.

Figure 9.15 shows the rules for detecting congestions induced by ICL. The second rule is the same rule found by Progol, while the first one imposed condition on three different sections of road. Also in this case the rules obtained by Progol and ICL are more precise than the rules obtained by ECL, but the recall is lower.

Figure 9.16 shows the rules for non–critical sections found by ECL. The first rule imposes conditions on all the three readings regarding the sections being described. The second rule imposes a condition also on the level of saturation

```
noncs(X,T)← velocidad(T,X,V_X),saturacion(T,X,S_X),
            ocupacion(T,X,O_X),V_X > 30.375,S_X > 73.125,
            O_X ≤ 828.625.                                  [0,0,34]
noncs(X,T)← velocidad(T,X,V_X),saturacion(T,Y,S_Y),
            saturacion(T,X,S_X),secciones_posteriores(Y,X),
            S_Y > 76.25V_X > 40.75,S_X ≤ 78.375.            [0,0,14]
noncs(X,T)← ocupacion(T,Y,O_Y),velocidad(T,X,V_X),
            ocupacion(T,X,O_X),secciones_posteriores(Y,X),
            O_Y ≤ 586.875,V_X > 30.375,O_X ≤ 748.125.       [1,0,72]
noncs(X,T)← ocupacion(T,X,O_X),tipo(X,rampa_incorporacion),
            saturacion(T,X,S_X),
            O_X ≤ 527.25,S_X ≤ 77.25.                       [0,1,15]
noncs(X,T)← ocupacion(T,Y,O_Y),velocidad(T,Y,V_Y),
            ocupacion(T,X,O_X),secciones_posteriores(Y,X),
            <O_Y ≤ 766.75,V_Y > 30.375,O_X ≤ 766.75.        [1,3,100]
noncs(X,T)← saturacion(T,Y,S_Y),velocidad(T,Y,V_Y),
            tipo(X,rampa_abandono),saturacion(T,X,S_X),
            ocupacion(T,X,O_X),secciones_posteriores(Y,X),
            S_Y ≤ 78.375,40.75 < V_Y ≤ 111.5,
            S_X ≤ 77.25,O_X ≤ 527.25.                       [0,2,11]
```

Figure 9.16: Rules for the non–critical class obtained by ECL.

of a previous section. In all the cases where the speed is involved, the average values imposed by the rules is high. This is confirmed by the rules obtained by Progol and ICL, shown in figure 9.17 and 9.18, respectively.

```
noncs(X,T) ←velocidad(T,X,alta),saturacion(T,X,media).       [0,0,32]
noncs(X,T)← saturacion(T,X,alta),tipo(X,carretera).          [0,0,30]
noncs(X,T)← secciones_posteriores(Y,X),ocupacion(T,X,baja).  [0,0,25]
noncs(X,T)← ocupacion(T,X,baja), saturacion(T,X,baja),
            tipo(X,rampa_incorporacion).                     [0,0,8]
noncs(X,T)← ocupacion(T,X,baja),secciones_posteriores(X,Y),
            saturacion(T,Y,media).                           [0,0,5]
noncs(X,T)← ocupacion(T,X,baja),secciones_posteriores(Y,X),
            saturacion(T,Y,alta).                            [0,0,11]
noncs(X,T)← ocupacion(T,X,baja),saturacion(T,X,baja),
            secciones_posteriores(Y,X),ocupacion(T,Y,media). [0,0,11]
noncs(X,T)← secciones_posteriores(Y,X),saturacion(T,Y,alta),
            secciones_posteriores(Y,Z),velocidad(T,Z,media). [0,0,12]
```

Figure 9.17: The set of rules generated by Progol for non–critical sections.

Validation

In this section we present the performance of the three systems as estimated by ten–fold cross validation. We also present the results obtained by Tilde and by C4.5.

The trees produced by Tilde were not included in the previous section be-

cause they are too difficult to interpret. We also include the results of C4.5 for veryfing if relational information was really needed in this problem. Being C4.5 a propositional system, it was run using only the sensor value for the focused section, and could not access values of neighboring sections.

Table 9.5 reports the results obtained by the various systems. In the last row of the table, results obtained by ECL using the global discretization proposed in the original study are reported. The reason for including these results is to verify if ECL can take advantage of the discretization established by experts or if the local supervised discretization method can find discretization intervals that yield better results in ECL.

```
class(noncs)←section(X),timemoment(T),velocidad(T,X,alta),
             secciones_posteriores(Y,X),velocidad(T,X,alta).  [0,0,46]
class(noncs)←section(X),timemoment(T),ocupacion(T,X,baja),
             secciones_posteriores(Y,X),velocidad(T,Y,alta),
             secciones_posteriores(Z,Y),velocidad(T,Z,alta).  [0,0,17]
class(noncs)←section(X),timemoment(T),tipo(X,carretera),
             secciones_posteriores(X,Y),ocupacion(T,Y,alta),
             secciones_posteriores(Y,Z),ocupacion(T,Z,alta).  [0.0,26]
```

Figure 9.18: An incomplete listing of the rules for non–critical sections obtained by ICL.

System	Accuracy	Simplicity	Time
ICL	0.93 (0.04)	18	82s
Progol	**0.94 (0.03)**	13	27min
Tilde	**0.94 (0.04)**	12	28s
C4.5	0.88 (0.05)	14	20ms
ECL-LSDc	0.93 (0.02)	15	25min
ECL	0.90 (0.03)	20	24min

Table 9.5: Performance of the systems as estimated by ten–fold cross validation.

The first thing that can be notice is that C4.5 obtained the worst results in term of accuracy. This is explained by the fact that information about the road geometry was needed in this problem. Being this information relational, C4.5 could not exploit it. This also confirms that for some classes of problems an ILP approach is needed in order to obtain good results.

As far as the accuracy is concerned, the performance of the ILP systems are comparable. Progol and Tilde obtained a higher accuracy than the one obtained by ECL-LSDc but the standard deviation is higher too. Also the simplicity of the solutions found is comparable, being the solutions obtained by Tilde the simplest and the solution of ICL the one containing more rules. In this case the simplicity is defined as the average number of rules induced for the three classes.

The main drawback of ECL-LSDc and Progol is their computational cost. In fact ICL and Tilde are much faster. C4.5 is the best performing system in terms of computational time.

Another interesting result is that for ECL, the discretization estrablished by experts is not the best way for dealing with numerical values in this problem. Better results are obtained with a local discretization.

An important aspect in this kind of study is represented by the comprehensibility of the induced rules. In fact rules induced by Tilde could not be interpreted easily, while the rules induced by ICL, Progol and ECL are easy to understand, since expressed by Horn clauses.

9.2.3 Conclusion for the Second Case

In this second case study we have addressed a problem regarding the automatic acquisition of knowledge about traffic problems. Data describing different situations that can happen on a road network has been used in order to induce rules for idenfing critical sections of road, i.e., sections in which an accident or a congestion took place. Background knowledge on road geometry is present, requiring the use of ILP for this task. The data used was generated using a simulator, however it should be noted that the simulator is capable of producing very realistic data and has been calibrated using real–world information.

The first objective of the second case study was to demostrate with a practical application that the use of ILP is needed for some class of problems. The performance of C4.5 confirmed this. In fact the performance obtained by C4.5 was the worst. This is justified by the fact that C4.5 can not take advantage of relational information.

With this second case study we also wanted to analyze more thoroughly an example of real life application of ECL to a relational problem, and show the ability of ECL to exploit relational information for building good rules. Another reason that lead us to choose this particular problem, is that the problem is charachterized by the presence of both numerical and relational information, and both these features are important in order to solve the problem. This is proved by the fact that C4.5 could not find good solutions for this problem, as already discussed, and by the fact that when ECL used different discretizations for dealing with numerical values, it obtained results of different quality.

ECL obtained better results when the numerical values were treated with a local discretization rather than using a global discretization established by experts in the field. This suggest that also the other systems could benefit from a different discretization.

From the studies presented in this chapter, it emerged that the main drawback of ECL is the computational time required by the evolutionary process. However, computational time is not the most important factor in the kind of

problems tackled in this chapter (Eiben and Smith, 2003a). In fact the knowledge extracted from the datasets is then used by either human experts or other expert systems. Thus it does not matter if the amount of time employed for extracting this knowledge is high. What matters the most is that the rules extracted are accurate. The opposite case, where computational time is an important factor, is represented by repetitive problems. In these problems a solution to a particular problem has to be found many times. An example of repetitive problem is to estrablish a daily schedule for a domestic transportation firm. Such a schedule should contain a pick-up and delivery plan plus a route description. This schedule has not to be the best schedule, but must be obtained in a small amount of time, and has to be reasonably good.

Chapter 10

Conclusions

With this thesis we wanted to design a hybrid EA for solving ILP problems. To this aim, we introduced the hybrid evolutionary system ECL.

The objectives of this thesis were discussed in section 1.3 of the Introduction, and illustrated in figure 1.1. In the following, for each point listed in section 1.3, we describe what has been achieved.

Effectiveness

1. We wanted to incorporate an **optimization** phase based on ILP operators for optimizing individuals of the current population. To this end, we incorporated an optimization phase in ECL that follows the mutation phase. In this phase individuals can be refined by means of the repeated application of mutation operators. Mutation operators perform ILP like operations, e.g., changing a variable into a constant. We have experimentally established that the incorporation of the optimization phase increases the exploitation power of ECL, allowing the system to achieve better results.
2. In order to perform ILP like operations, we wanted to adopt a **representation** close to the Prolog syntax. ECL adopts a high level representation language, similar to the one adopted by SIA01. Clauses are encoded in ECL as a list of predicate symbols, variables and constants. This representation not only makes it possible to apply ILP oriented mutation operators, but has also another appealing aspect: with such a representation clauses are not required to follow a fixed form, given, e.g., as a user supplied template. The shape of each clause is determined by the example used as seed.
3. We wanted to develop **genetic operators** that bias the search toward better hypotheses. Four mutation operators are used to this end. These mutation operators are greedy because they consider a number of mutation possibilities. Each possibility is tested, and the one

yielding the best improvement in the fitness of the individual is applied. Results of experiments confirm that the use of greedy mutation operators is beneficial in order to achieve better results, both in terms of accuracy and simplicity.

No crossover operator is used. The reason behind this choice is that it is difficult to design an effective crossover operator with the high level representation adopted by ECL. Some experiments were conducted with a uniform crossover, but the results of such experiments did not justify its use.

4. We wanted to develop some mechanisms for promoting **diversity** in the population as well as good coverage of positive examples. Maintaining diversity is a key factor in the system, because the use of greedy mutation operators and the optimization phase can cause the population to converge to a number of super individuals.

   To this end, we use a selection operator that incorporates knowledge, by means of a weight assigned to each example. This weight determines the difficulty of the example. Weights are adjourned at each generation, and depend on the number of individuals covering the examples. Individuals are selected in two steps: first a number of positive examples are selected, then one individual for each selected example is selected for being mutated. The probability of selection of each example depends on its estimated difficulty. Difficult examples have a greater chance of being selected. In this way diversity in the population is promoted, as well as a good coverage of the examples.

   In this thesis, we have experimentally shown that having a good diversity in the population is positively reflected in the quality of the found solutions.

   We believe that maintaining a good diversity in the population is an important aspect in all EAs for ICL. We have seen that parallel systems, like REGAL and G–NET, promote diversity by means of a co–evolution policy promoted by a supervisor node. Our proposed method for promoting diversity relies only on the selection operator, and does not use any distance measure, which renders this strategy efficient. The method can be applied to any non-parallel EAs for ICL. This method recalls in some ways boosting (Schapire, 1990). In boosting what is done is to call several times a learning algorithm on a given subset of the training examples. At first, this subset coincides with the entire training set. After each call of the learner the composition of the subset is changed, and depends on the estimated difficulty of each example. This procedure is repeated until a theory (consisting of an ensemble of learners) of satisfactory quality has been found.

5. We wanted to introduce methods for handling **numerical values**. To this aim, we have introduced three methods for handling nu-

merical values in ECL. Many ICL problems are characterized by the high presence of numerical values. This holds also for ILP problems, therefore having a good method for dealing with numerical values becomes a key aspect. We believe that a global discretization does not represent the optimal solution in ICL, and in particular in ILP. A local supervised discretization seems to be a more appealing way for dealing with numerical values. The methods proposed in this thesis are successful in helping ECL achieving better results in problems characterized by a high presence of numerical values.

Efficiency

1. Incorporating knowledge in the system by means of greedy mutation operators implies an increment of the **computational time** required by the evolutionary process. This time increases again with the incorporation of the optimization phase. For this reason we have introduced a simple stochastic sampling of the background knowledge in reducing the computational effort required by evaluation of clauses. This method is different from other sampling methods that typically rely on a sampling of the training examples. Experiments show that even if a small amount of the background knowledge is used, ECL can find results of satisfactory quality.
2. We wanted to exploit the natural **parallelism** of GAs in order to reduce the computational time. To this end, we have implemented two parallel versions of ECL. We have shown that the two parallelizations are effective in order to reduce the computational time required by the learning process carried out by ECL. The parallel versions of ECL are less successful for improving the accuracy of the found solutions.

As far as effectiveness is concerned, we can state that we were successful in achieving our objectives. The performance of ECL is comparable or superior to the performance of other state of the art systems for ICL, both in the propositional and in the relational setting.

The two solutions adopted for incrementing the efficiency of ECL achieved their objectives. We can state that the efficiency of ECL represents its main weakness. However, for the kind of problems tackled in this thesis, computational time is not the main issue. In fact all the kind of problems we have tackled are not repetitive problems, where a solution of satisfactory quality has to be found very often and in a small amount of time (Eiben and Smith, 2003c).

Instead, in the class of problems addressed by ECL, the main issue is the effectiveness of the system. We want to obtain a solution that is as accurate as possible. Of course we would like to have our system to be as fast as possible. However, we are mainly interested in the accuracy of the results. Computational time can be reduced by having faster hardware, and hardware is becoming faster and faster everyday. The knowledge that is acquired is then used

by experts for some purposes, e.g., for the design of a expert system for controlling the traffic situation on a network of roads, as it happens in the second case study proposed in chapter 9. In such cases we are interested mainly in the quality of the solutions found: we prefer to have an accurate solution obtained with a high computational time rather then a solution of lower quality obtained in a small amount of time.

10.1 Future Work

Several parameters are used, and must be tuned, for controlling various aspects of the evolutionary process performed by ECL. In order to find a good setting of these parameters, many preliminary runs on training examples have to be executed with different values of the parameters. This is a time consuming operation. Besides it is not guaranteed that in this way the optimal setting of parameters is found. In the future a line of research that we intend to address, regards the development of some self tuning methods for the automatic tuning of some of these parameters. For instance, the parameters N_i used for controlling each mutation operators can be varied during the evolutionary process. Low degrees of greediness can be used when the individual to be mutated is characterized by a poor fitness, while more greediness can be used when the fitness of the individual is considered good. In this way we give more exploration "power" to the operators when the individuals are not good, while individuals will be refined when they are considered to be already good. Another parameter that may be self adapted is pbk, used for sampling the background knowledge. Different iterations may use different values of pbk in order to use more or less background knowledge. The values of pbk could be determined considering the quality of the population evolved in the previous iteration.

In the version of ECL introduced by this thesis, no crossover operator is used. Another future development is to design an effective crossover operator that can be used with the high level representation adopted by ECL. This would help the system in maintaining diversity in the population.

In the actual version of ECL the background knowledge consists of a set of ground facts, so it is an extensional background knowledge. An extensional background knowledge can be generated from an intensional background BK' by generating all ground facts derivable from BK' in at most h resolution steps, where the value of h can be provided by the user. This has the limitation of allowing the system to use only facts that can be obtained with at most h derivation steps. Another limitation is the fact that in this way the background knowledge is not as compact as an intensional background knowledge. Having an intensional background knowledge allows to exploit some a priori known structural properties. In the future we intend to extend ECL in this direction, thus allowing the system to use also theories in the background knowledge.

Another limitation of ECL is its incapability of directly addressing multi-

class problems. At the moment, these problems are tackled by learning a theory for each class, using the positive examples of other classes as negative examples of the class subject to the learning process. In the future, we also intend to extend ECL in order to render it able to deal with this kind of problems.

We have seen that ECL-LUD can not exploit class information of examples, and this renders the method less successful than the supervised discretization methods. A possible variant of ECL-LUD could be designed, by changing the way in which the operators adopted for modifying inequalities act, in order to exploit information on the class of examples, turning in this way ECL-LUD into a supervised discretization method. This could be done, e.g., by estimating the density distribution of positive and negative examples inside each cluster and then use this information when modifying inequalities.

An interesting development of ECL would be its extension to Constraint Logic Programming (CLP) (Cohen, 1996; Jaffar and Maher, 1994). CLP programs are LP programs in which unification is replaced by constraint solving in various domains. Constraints are special predicates whose satisfiability can be established for various domains using different algorithms. Unification can be seen as a particular type of constraint. As a result, CLP is more powerful than LP. It would be interesting to extend ECL in order to induce CLP programs instead of LP programs.

The parallel version of ECL needs many developments. The present implementation is successful in reducing the computational time required by the learning process, but it is less successful for what concerns the accuracy of the solution found. As a future development in this direction, we intend to investigate a way of migrating portion of the background knowledge among the nodes by means of a co-evolution policy promoted by a supervisor node. By changing the portion of the background knowledge assigned to each node, the genetic search performed on the node is shifted toward other regions of the hypothesis space. Other lines of investigations may regard different communication topologies.

Acknowledgments

This thesis is the results of four years of "hard" work at the Vrije Universiteit of Amsterdam. I really enjoyed my time here. If I look back in these years, I can think of a lot of people I want to thank. I want to thank people for supporting and helping me in my research project, for their friendship, for the moral support they provided me and for simply being there.

I am grateful to Elena Marchiori, my academic supervisor. In these four years you thought me valuable lessons on how to do research and writing scientific articles. Not only this, you also gave me a sense of direction and focus whenever I needed it.

During my second year of Ph.D., Maarten Keijzer started to work in our group. Maarten, you have really helped me a lot in my Ph.D., you gave me a big push for finishing this thesis. Many thanks go to my promotor, Gusz Eiben. It was a pleasure and a honor to work in your group. I also want to thank Wojtek Kowalczyk and Bart Craenen. You gave a great contribution to render my stay at the VU very pleasant. Thanks go to Sandjai Bhulai. Whenever I had some questions you were always willing to help me out. Thanks to Auke Pot as well, I really enjoyed sharing the office with you.

Then I would like to thank Raquel. I really liked my work here, but I can affirm that meeting you was the best thing that could have occured to me. Even in the cloudiest day you could bring the sun light in my life: *ẽre-lo meu sol e o meu ceo*.

I also want to thank my family. Vi ringrazio tutti, mamma, Paola, Nadia, Fulvio, Remo, Paolo, Bepi, Claudia, Roberta grande e Roberta piccola, Simone, Marco, Chiara, Martina e Giovanni. Potete anche essere lontani in distanza, ma siete sempre vicini a me nel mio cuore. Voglio anche ringraziare mio padre. Papá, so che tu mi guardi sempre da lassú. Un ringraziamento va anche a Gino Dalle Fratte per avermi consigliato di iniziare il dottorato.

Since my arrival in the Netherlands I met many people that became my friends. I would like to thank all these people for their precious friendship. Alessandro, your friendship has been, and is, very important to me. We really had a good time, and many beers at the Belgian pub. They should, at least, dedicate a table to us there. Radu, you have been one of the first friends I made here, and a very good one, I must say. Giovanni, you are such a good friend, and such a great cook! I also want to thank for their friendship Bogdan,

Spyros, Carmelo, Luca , Aarno, Sonia, Berenice, Idoia, Jota, Cora, Maria and Marta. Other friends are in Italy, but are nonetheless important to me: Ivan, Annarita, Andrea, Roy, Erika, Fritz, Fabrizio and Luigi.

I am also grateful to Sophie Kain, Dawids Edward and Peter Bentley for evolving with me nice flying objects.

I am grateful to Jesús Aguilar-Ruiz and Jaume Bacardit, for the nice work we have done together. Thanks also to Raúl Giráldez for his help in running some experiments contained in this thesis. I also remember with pleasure the nice discussion with Michèle Sebag.

Many thanks to Ioan Staicu for his work regarding the parallelization of ECL.

I have some many people to be grateful to, that I have probably forgotten to mention some of them. All of you have really helped me a lot in these four years: grazie!

Samenvatting

Dit proefschrift heeft als onderwerp "Hybride Evolutionaire Computatie voor Inductief Leren". Het centrale thema van dit proefschrift is het gebruik van hybride Evolutionaire Computatie (EC) voor Inductief Logisch Programmeren (ILP).

EC is een stochastische methode, gebaseerd op een populatie, voor het oplossen van optimalisatie problemen. EC is in beginsel gebaseerd op de evolutietheorie van Darwin. Kandidaat-oplossingen worden door selectie, kruising en mutatie geëvolueerd. Het idee is dat na elke stap betere oplossingen worden verkregen.

ILP bevindt zich op de doorsnede van inductief leren en logisch programmeren. ILP erft haar belangrijkste doelstelling van het inductief leren, namelijk het leren van theorieën aan de hand van voorbeelden. Van logisch programmeren wordt het formalisme van kennisrepresentatie overgenomen: Horn clause logica. De klassieke oplossingsmethode voor ILP problemen is het gebruik van local-search technieken, de zogeheten gretige zoektechnieken. Deze technieken kunnen snel redelijk goede oplossingen verbeteren, maar hebben als nadeel dat ze in lokale optima kunnen blijven steken.

EC is een techniek om goede oplossingen te vinden, maar werkt minder goed om deze te verbeteren. De combinatie van klassieke methoden met EC kan profiteren van de goede kenmerken van beide technieken. Dit is dan ook de belangrijkste motivatie voor de introductie van het ECL systeem.

Het eerste hoofdstuk bevat de introductie van dit proefschrift. Hoofdstuk 2 bevat een introductie tot de basisbegrippen van ILP. De eerste secties van dit hoofdstuk gaan over de representatie in Horn clause logica. Het hoofdstuk sluit af met een beschrijving van twee systemen voor ILP, namelijk FOIL en Prolog.

Hoofdstuk 3 gaat over de basisbeginselen van EC. Alle EC aspecten die nodig zijn voor het begrijpen van dit proefschrift worden hierin beschreven. De noodzaak om diversiteit wordt te handhaven in de populatie uitgelegd. Het hoofdstuk eindigt met een beschrijving van hybride EC.

Hoofdstuk 4 bevat een beschrijving van vijf evolutionaire systemen voor ILP, namelijk REGAL, G-NET, DOGMA, SIA01 en GLPS. Deze systemen hebben uiteenlopende kenmerken: ze gebruiken allemaal selectie, kruising, mutatie, een fitness functie en representatie. Deze worden in het hoofdstuk beschreven.

Het hoofdstuk sluit af met een discussie over de kenmerken van de vijf systemen.

ECL wordt in hoofdstuk 5 beschreven. Eerst wordt de motivatie voor de introductie van ECL gegeven. Daarna worden de kenmerken van ECL beschreven. De belangrijkste kenmerken van ECL zijn het gebruik van een optimalisatie fase welke de mutatie fase opvolgt, het gebruik van intelligente mutatie operatoren en een selectie operator die de diversiteit in de populatie bevordert.

De methoden die door ECL worden gebruikt voor numerieke attributen worden in hoofdstuk 6 beschreven. Drie methoden worden geïntroduceerd. De eerste methode kan informatie over de aard van de voorbeelden niet gebruiken, terwijl de andere twee methoden dat wel kunnen. Het hoofdstuk sluit af met een beschrijving van andere methoden om met numerieke attributen om te gaan.

Een experimentele evaluatie van de componenten van ECL wordt in hoofdstuk 7 gepresenteerd. In dit hoofdstuk wordt er ook een vergelijking van ECL met andere systemen gegeven. Het resultaat van deze evaluatie is dat ECL vergelijkbare of betere resultaten behaalt dan andere systemen.

Twee parallelle versies van ECL worden in hoofdstuk 8 beschreven. Deze versies zijn geschikt om de benodigde tijd voor het evolutionaire proces te verminderen. Ze zijn minder geschikt om de nauwkeurigheid van de oplossingen te verbeteren.

Hoofdstuk 9 gaat over de beschrijving van twee case studies. De eerste case studie gaat over de relatie tussen artsen en patiënten. De tweede case studie gaat over het opsporen van verkeersproblemen, zoals files.

Conclusies worden tenslotte in hoofdstuk 10 gegeven.

Bibliography

Aguilar-Ruiz, J. S., Costa, R., and Divina, F. (2004). Knowledge discovery from doctor-patient relationship. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 280–284. ACM Press.

Aguilar-Ruiz, J. S., Riquelme, J. C., and Toro, M. (2003). Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 33(2):324–331.

Aguilar-Ruiz, J. S., Riquelme, J. C., and Valle, C. D. (2002). Improving the evolutionary coding for machine learning tasks. In *European Concefence on Artificial Intelligence, ECAI'02*, pages 173–177, Lyon, France. IOS Press.

Aha, D. W., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

Anglano, C., Giordana, A., Bello, G. L., and Saitta, L. (1998). An experimental evaluation of coevolutive concept learning. In *Proc. 15th International Conf. on Machine Learning*, pages 19–27. Morgan Kaufmann, San Francisco, CA.

Augier, S., Venturini, G., and Kodratoff, Y. (1995). Learning first order logic rules with a genetic algorithm. In Fayyad, U. M. and Uthurusamy, R., editors, *The First International Conference on Knowledge Discovery and Data Mining*, pages 21–26, Montreal, Canada. AAAI Press.

Bacardit, J. and Garrel, J. M. (2002). Evolution of multi-adaptive discretization intervals for a rule-based genetic learning system. In *Proceedings of the 7th Iberoamerican Conference on Artificial Intelligence (IBERAMIA2002)*, pages 350–360.

Bacardit, J. and Garrel, J. M. (2003). Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based genetic learning classifier system. In *GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1818–1831. Springer.

Bacardit, J. and Garrell, J. M. (2003). 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.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (2000a). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (2000b). *Evolutionary Computation 2: Advance Algorithms and Operators*. Institute of Physics Publishing.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, Publishers.

Barker, D. A., Shergill, S. S., Higginson, I., and Orrel, M. W. (1996). Patients' views towards care received from psychiatrists. *British Journal of Psychiatry*, 168:641–646.

Bentley, P. J. (1999). *Evolutionary Design by Computers*. Morgan Kaufmann Publishers Inc.

Bentley, P. J. and Corne, D. W. (2001). *Creative evolutionary systems*. Morgan Kaufmann Publishers Inc.

Bhoedjang, R., Ruhl, T., and Bal, H. (2003). User-level network interface protocols. *IEEE Computer*, 31(11):53–60.

Blake, C. and Merz, C. (1998). UCI repository of machine learning databases.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical Report 11, Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland.

Blockeel, H. and De Raedt, L. (1997). Lookahead and discretization in ilp. In Džeroski, S. and Lavrač, N., editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 77–84. Springer-Verlag.

Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Mon, J., and Vandecasteele, H. (2002). Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166.

Blockeel, H. and Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, 24(6):377–380.

Cantu-Paz, E. and Kamath, C. (2003). 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 Kodratoff, Y., editor, *European Working Session on Learning*. Springer-Verlag. LNAI 482.

Clark, P. and Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proc. Fifth European Working Session on Learning*, pages 151–163, Berlin. Springer.

Cohen, J. (1996). Logic programming and constraint logic programming. *ACM Comput. Surv.*, 28(1):257–259.

Corne, D., Ross, P., and Fang, H.-L. (1994). Fast practical evolutionary timetabling. In *Lecture Notes in Computer Science, vol 865 (Evolutionary Computing AISB Workshop, Leeds, UK, April 1994)*, pages 251–263. Springer-Verlag.

Costa, R. and Divina, F. (2003a). Application of inductive concept learning to doctor–patient relation data. In *BNAIC 2003: Proceedings of the Belgian-Dutch Conference on Artificial Intelligence*, pages 67–74.

Costa, R. and Divina, F. (2003b). Doctor-patient relation dataset (http://www.cs.vu.nl/∼divina).

Cuena, J., Hernandez, J., and Molina, M. (1995). Knowledge-based models for adaptive traffic management systems. *Transportation Research: Part C*, 3(5):311–337.

Darwin, C. (1859). On the origin of the species by means of natural selection, or the preservation of favoured races in the struggle for life.

Davis, L. (1985). Job shop scheduling with genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 136–140, Mahawah, NJ. Lawrence Eribaum Associates.

De Jong, K. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI.

De Jong, K., Spears, W., and Gordon, D. (1993). Using Genetic Algorithms for Concept Learning. *Machine Learning*, 13(1/2):155–188.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92.

De Raedt, L. and Van Laer, W. (1995). Inductive constraint logic. In *Proceedings of the 6th Conference on Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Debnath, A., de Compadre, R. L., Debnath, G., Schusterman, A., and Hansch, C. (1991). Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medical Chemistry*, 34(2):786–797.

DeJong, K. A. and Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 651–656.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the the Royal Statistical Society*, 39:1–38.

Divina, F. (2001a). Evolutionary concept learning in first order logic. Technical Report IR-494, Vrije Universiteit Amsterdam.

Divina, F. (2001b). Knowledge based evolutionary computing for inductive learning in first-order logic. In *Proceedings of the Brussels Evolutionary Algorithms Day (BEAD-2001), Workshop on EAs*, pages 10–15, Brussels.

Divina, F., Edwards, D., and Kain, S. (2003a). Creative evolution of flying objects. In *Proceedings of CCIA*, pages 276–285, Palma de Mallorca. IOS-Press.

Divina, F., Keijzer, M., and Marchiori, E. (2002). Non-universal suffrage selection operators favor population diversity in genetic algorithms. In *Benelearn 2002: Proceedings of the 12th Belgian-Dutch Conference on Machine Learning (Technical report UU-CS-2002-046)*, pages 23–30, Utrecht, The Netherlands.

Divina, F., Keijzer, M., and Marchiori, E. (2003b). Evolutionary concept learning with constraints for numerical attributes. In *BNAIC 2003: Proceedings of the Belgian-Dutch Conference on Artificial Intelligence*, pages 107–114, Nijmegen, The Netherlands.

Divina, F., Keijzer, M., and Marchiori, E. (2003c). A method for handling numerical attributes in GA-based inductive concept learners. In et al., E. C.-P., editor, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 898–908, Chicago. Springer-Verlag.

Divina, F. and Marchiori, E. (2001). Knowledge based evolutionary programming for inductive learning in first-order logic. In et al., L. S., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 173. Morgan Kaufmann.

Divina, F. and Marchiori, E. (2002). Evolutionary concept learning. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 343–350, New York. Morgan Kaufmann Publishers.

Divina, F. and Marchiori, E. (2004a). Handling continuous attributes in an evolutionary inductive learner. *IEEE Transactions on Evolutionary Computation*, to appear.

Divina, F. and Marchiori, E. (2004b). Knowledge–based evolutionary search for inductive concept learning. In *Knowledge Incorporation in Evolutionary Computation*, chapter III, pages 237–254. Springer-Verlag.

Dolšak, B., Bratko, I., and Jezernik, A. (1994). Finite element mesh design: An engineering domain for ILP application. In Wrobel, S., editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 305–320. Gesellschaft für Mathematik und Datenverarbeitung MBH.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202.

DSM-IV (2000). *Diagnostic criteria from DSM-IV TR*. American Psychiatric Association, Washington DC.

Džeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., and Laer, W. V. (1999). Experiments in predicting biodegradability. In *International Workshop on Inductive Logic Programming*, pages 80–91.

Džeroski, S., Jacobs, N., Molina, M., and Moure, C. (1998a). ILP experiments in detecting traffic problems. In *European Conference on Machine Learning*, pages 61–66.

Džeroski, S., Jacobs, N., Molina, M., Moure, C., Muggleton, S., and Laer, W. V. (1998b). Detecting traffic problems with ILP. In *International Workshop on Inductive Logic Programming*, pages 281–290.

Eiben, A. E. and Smith, J. E. (2003a). *Introduction to Evolutionary Computing*. Springer-Verlag.

Eiben, A. E. and Smith, J. E. (2003b). What is an evolutionary algorithm? In *Introduction to Evolutionary Computing*, chapter 2, pages 15–35. Springer-Verlag.

Eiben, A. E. and Smith, J. E. (2003c). Working with evolutionary algorithms. In *Introduction to Evolutionary Computing*, chapter 14, pages 243–262. Springer-Verlag.

Fayyad, U. and Irani, K. (1992). On the handling of continuous-valued attributes in decision tree generation. *Mach. Learn.*, 8:87–102.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuos attributes as pre-processing for classification learning. In *Proceedings of the 13th International Join Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann Publishers.

Ferrer, J. and Barcelo, J. (1993). AIMSUN2: Advanced Interactive Microscopic Simulator for Urban and non-urban Networks. Technical report, Departamento de Estad istica e Investigacion Operativa, Facultad de Informatica, Universitat Politecnica de Catalynya.

Fogel, D. B. (1988). An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 6:2:139–144.

Fogel, D. B. (1993). Evolving behavious in the iterated prisoner's dilemma. *Evolutionary Computation*, 1(1):77–97.

Fogel, L. J., Owens, A. J., and Walsh, M. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley.

Freitas, A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Spinger-Verlag, Berlin.

Freitas, A. and Lavington, S. (1998). *Mining very large databases with parallel processing*. Kluwer.

Gehlhaar, D. K., Verkhivker, G. M., Rejto, P. A., Sherman, C. J., Fogel, D. B., Fogel, L. J., , and Freer, S. T. (1995). Molecular recognition of the inhibitor ag-1343 by hiv-1 protease: conformationally flexible docking by evolutionary programming. *Chemistry and Biology*, 2(5):317–324.

Giordana, A. and Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416.

Giordana, A. and Saitta, L. (2000). Phase Transitions in Relational Learning. *Machine Learning*, 41(2):217–251.

Giráldez, R. (2004). *Mejoras en Eficiencia y Eficacia de Algoritmos Evolutivos para Aprendizaje Supervisado*. PhD thesis, Universidad de Sevilla.

Giráldez, R., Aguilar-Ruiz, J. S., and Riquelme, J. C. (2003). Natural coding: A more efficient representation for evolutionary learning. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 979–990, Chicago. Springer-Verlag.

Giráldez, R., Aguilar-Ruiz, J. S., and Riquelme, J. C. (2004). Knowledge-based fast evaluation for evolutionary learning. *IEEE Transactions on Systems, Man & Cybernetics: Part C – Special Issue on Knowledge Extraction and Incorporation in Evolutionary Computation*, (in press).

Giráldez, R., Aguilar-Ruiz, J. S., Riquelme, J. C., Ferrer-Troyano, F., and Rodriguez, D. (2002). Discretization oriented to decision rules generation. *Frontiers in Artificial Intelligence and Applications*, 82:275–279.

Glover, R. and Sharpe, P. (1999). Efficient GA based techniques classification. *Applied Intelligence*, 11(3):277–289.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann Publishers.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proc. of 2nd Int'l Conf. on Genetic Algorithms*, pages 41–49. Morgan Kaufmann Publishers.

Goldberg, D. E. and Robert, L. (1985). Alleles, loci and the travelling salesman problem. In Grefenstette, J. e., editor, *Proceedings of 1st Int. Conf. on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, Hillsdale.

Hekanaho, J. (1996). Background knowledge in GA-based concept learning. In *International Conference on Machine Learning*, pages 234–242.

Hekanaho, J. (1998). DOGMA: a GA based relational learner. In Page, D., editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, LNAI 1446, pages 205–214. Springer Verlag.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Holland, J. H. (1987). Genetic algorithms and classifier systems: foundations and future directions. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Mach. Learn.*, 11(1):63–90.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581.

Jakob, W., Gorges-Schleuter, M., and Blume, C. (1992). Application of Genetic Algorithms to task planning and learning. In Mánner, R. and Manderick, B., editors, *Parallel Problem Solving from Nature, 2nd Workshop*, Lecture Notes in Computer Science, pages 291–300, Amsterdam. North-Holland Publishing Company.

Janikow, C. (1993). A knowledge intensive genetic algorithm for supervised learning. *Machine Learning*, 13:198–228.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345.

Kaplan, H. I., Sadock, B. J., and Grebb, J. A. (1994). *Synopsis of Psychiatry*. Williams and Wilkins.

Keijzer, M. (2002). *Scientific Discovery Using Genetic Programming*. PhD thesis, Danish Technical University, Lyngby, Denmark.

Kerber, R. (1992). ChiMerge: discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 123–127. AAAI Press.

King, R. D., Sternberg, M. J. E., and Srinivasan, A. (1995). Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing*, 13(3&4):411–433.

Kohavi, R. and Sahami, M. (1996). Error-based and entropy-based discretization of continuous features. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 114–119.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA.

KrishnaKumar, K. and Goldberg, D. E. (1990). Genetic algorithms in control system optimization. *AIAA Guidance, Navigation and Control Conference, Portland*.

Kubat, M., Bratko, I., and Michalski, R. (1998). A review of Machine Learning Methods. In Michalski, R., Bratko, I., and Kubat, M., editors, *Machine Learning and Data Mining*. John Wiley and Sons Ltd., Chichester.

Kwedlo, W. and Kretowski, M. (1999). An evolutionary algorithm using multivariate discretization for decision rule induction. In *Principles of Data Mining and Knowledge Discovery*, pages 392–397.

Likert, R. (1932). *Technique for the Measurement of Attitudes*. PhD Thesis of Columbia University.

Liu, H., Hussain, F., Tan, C., and Dash, M. (2002). Discretization: An enabling technique. *Journal of Data Mining and Knowledge Discovery*, 6(4):393–423.

Llorá, X. and Garrell, J. M. (2001a). Inducing partially-defined instances with evolutionary algorithms. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 337–344. Morgan Kaufmann Publishers Inc.

Llorá, X. and Garrell, J. M. (2001b). Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In et al., L. S., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 461–468. Morgan Kaufmann.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume I, pages 281–297. University of California Press, Berkeley and Los Angeles.

Marchiori, E. and Steenbeek, A. (2000). An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling. In *Real World Applications of Evolutionary Computing. Springer*, pages 367–381. Springer-Verlag.

Michalski, R., Carbonell, J., and Mitchell, T. (1983). A theory and methodology of inductive learning. In *Machine Learning: An AI Approach*, volume I, pages 83–134. Morgan Kaufmann, Los Altos, CA.

Michalski, R., Mozetic, I., Hong, J., and Lavrač, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. Fifth National Conference on Artificial Intelligence*, pages 1041–1045. American Association for Artificial Intelligence (Philadelphia, PA).

Mira, J. J. (2001). La satisfaccion del paciente.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

Mitchell, T. (1997). *Machine Learning*. Series in Computer Science. McGraw-Hill.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program, Californian Institute of Technology, U.S.A.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Progra mming*, 13(3-4):245–286.

Muggleton, S. (1996). Learning from positive data. In Muggleton, S., editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 225–244. Stockholm University, Royal Institute of Technology.

Muggleton, S. (1999). Inductive logic programming: issues, results and the challenge of learning language in logic. *Artificial Intelligence*, 114:283–296.

Muggleton, S. and Buntine, W. (1988). Machine invention of first order predicates by inverting resolution. In *Proceedings of the Fifth International Machine Learning Conference*, pages 339–351. Morgan Kaufmann.

Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19-20:669–679.

Neri, F. and Saitta, L. (1995). Analysis of genetic algorithms evolution under pure selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 32–39. Morgan Kaufmann, San Francisco, CA.

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods: support vector learning*, pages 185–208.

Quinlan, J. (1990). Learning logical definition from relations. *Machine Learning*, 5:239–266.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Machine Learning. Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*. Morgan Kaufmann. Originally published in *Machine Learning* 1:81–106, 1986.

Quinlan, J. R. (1998-2001). See5.0 (http://www.rulequest.com).

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Friedrich Frommann.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry*. World Scientific, River Edge, NJ.

Roosen, P. and Meyer, F. (1992). Determination of chemical equilibria by means of an evolution strategy. In Männer, R. and Manderick, B., editors, *Parallel problem solving from nature 2*, pages 411–420, Amsterdam. North-Holland.

Ruggeri, M. and Dall'Agnola, R. (1993). Verona service satisfaction scale (vsss-54).

Russel, S. and Norvig, P. (1995). *Artificial Intelligence - A modern approch*. Englewood Cliffs, NJ: Prentice-Hall.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197–227.

Sebag, M. and Rouveirol, C. (1997). Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann.

Spencer, G. F. (1993). Automatic generation of programs for crawling and walking. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, page 654, University of Illinois at Urbana-Champaign. Morgan Kaufmann.

Staicu, I. (2003). A parallel genetic algorithm used for inductive concept learning. Master's thesis, Vrije Universiteit, Amsterdam.

Stone, C. and Bull, L. (2003). For real! xcs with continuous-valued inputs. *Evolutionary Computation Journal*, 11(3):298–336.

Tanaka, Y., Ishiguro, A., and Uchikawa, Y. (1993). A genetic algorithms' application to inverse problems in electromagnetics. In Forrest, S., editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, page 656, San Mateo, CA. Morgan Kaufmann.

Teller, A. and Andre, D. (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA. Morgan Kaufmann.

Van Laer, W. (2002). *From Propositional to First Order Logic in Machine Learning and Data Mining - Induction of first order rules with ICL*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium. 239+xviii pages.

Weingarten, S. R., Stone, E., Green, A., Pelter, M., Nessim, S., Huang, H., and Kristopaitis, R. (1995). A study of patient satisfaction and adherence to preventive care practice guidelines. *The American Journal of Medicine*, 99:590–595.

Whitley, D., Rana, S. B., and Heckendorn, R. B. (1997). Island model genetic algorithms and linearly separable problems. In *Evolutionary Computing, AISB Workshop*, pages 109–125.

Wilson, S. W. (1998). Generalization in the XCS classifier system. In Koza, J. R. and et. al., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

Witten, I. H. and Frank, E. (2000a). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers.

Witten, I. H. and Frank, E. (2000b). *WEKA Machine Learning Algorithms in Java*, chapter 8, pages 265–320. Morgan Kaufmann Publishers.

Wong, M. L. and Leung, K. S. (1995). Inducing logic programs with genetic algorithms: The genetic logic programming system. In *IEEE Expert 10(5)*, pages 68–76.

Yamada, T. and Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems. In R. Männer and Manderick, B., editors, *Parallel Problem Solving from Nature, 2*, Amsterdam. Elsevier Science Publishers, B. V.

Yao, X. (2002). *Evolutionary computation: A gentle introduction*, chapter 2, pages 27–53. Kluwer Academic Publishers.

Index

SIKS Dissertatiereeks

1998

1998-1 Johan van den Akker (CWI) DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM) Memory versus Search in Games

1998-5 E.W.Oskamp (RUL) Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR) Classification using decision trees and neural nets

1999-3 Don Beal (UM) The Nature of Minimax Search

1999-4 Jacques Penders (UM) The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU) Re-design of compositional systems

1999-7 David Spelt (UT) Verification support for object database design

1999-8 Jacques H.J. Lenting (UM) Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-1 Frank Niessink (VU) Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE) Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA) Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU) ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM) Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU) Programming Languages for Agent Communication

2000-7 Niels Peek (UU) Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR) Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI) Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI) Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI) Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU) Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU) Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA) Learning as problem solving

2001-4 Evgueni Smirnov (UM) Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU) Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU) Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU) Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU) A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL) Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA) Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA) Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-1 Nico Lassing (VU) Architecture-Level Modifiability Analysis

2002-2 Roelof van Zwol (UT) Modelling and searching web-based document collections

2002-3 Henk Ernst Blok (UT) Database Optimization Aspects for Information Retrieval

2002-4 Juan Roberto Castelo Valdueza (UU) The Discrete Acyclic Digraph Markov Model in Data Mining

2002-5 Radu Serban (VU) The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-6 Laurens Mommers (UL) Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-7 Peter Boncz (CWI) Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-8 Jaap Gordijn (VU) Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-9 Willem-Jan van den Heuvel(KUB) Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM) Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU) Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva) Processing XML in Database Systems

2002-13 Hongjing Wu (TUE) A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU) Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT) Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU) The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA) Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-1 Heiner Stuckenschmidt (VU) Ontology-Based Information Sharing in Weakly Structured Environments

2003-2 Jan Broersen (VU) Modal Action Logics for Reasoning About Reactive Systems

2003-3 Martijn Schuemie (TUD) Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-4 Milan Petkovic (UT) Content-Based Video Retrieval Supported by Database Technology

2003-5 Jos Lehmann (UVA) Causation in Artificial Intelligence and Law - A modelling approach

2003-6 Boris van Schooten (UT) Development and specification of virtual environments

2003-7 Machiel Jansen (UvA) Formal Explorations of Knowledge Intensive Tasks

2003-8 Yongping Ran (UM) Repair Based Scheduling

2003-9 Rens Kortmann (UM) The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT) Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT) Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT) Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM) Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN) Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD) Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI) Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT) Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM) Learning Search Decisions

2004

2004-1 Virginia Dignum (UU) A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-2 Lai Xu (UvT) Monitoring Multi-party Contracts for E-business

2004-3 Perry Groot (VU) A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-4 Chris van Aart (UVA) Organizational Principles for Multi-Agent Architectures

2004-5 Viara Popova (EUR) Knowledge discovery and monotonicity

2004-6 Bart-Jan Hommes (TUD) The Evaluation of Business Process Modeling Techniques

2004-7 Elise Boltjes (UM) Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-8 Joop Verbeek(UM) Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-9 Martin Caminada (VU) For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA) Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU) Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT) Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT) Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU) Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU) Multi-Relational Data Mining