An Evolutionary Approach to Concept Learning

Jukkapekka Hekanaho

Department of Computer Science
Åbo Akademi University

ISBN 952-12-0343-9
ISSN 1239-1883

Painosalama Oy

Acknowledgements

As my thesis is now written, it is time to pause for a while, to take a look back and thank all the people who have been involved in the process. I would like to start by thanking my supervisor Professor Ralph Back. He has had long-sightedness in allowing me to do what I wanted to do and has had patience with the long incubation of my thesis.

Next, my thanks go to the whole machine learning group at the University of Turin with whom I spent a year in 1994-95. Especially, I would like to thank Professors Attilio Giordana and Lorenza Saitta who kindly agreed to let me stay in their group. It is during this period that I started the work that finally led to this thesis.

I should also thank a few biochemists. Anders Aspnäs and Vic Cockcroft are co-authors of one of the papers underlying this thesis. With them, as well as with Professor Mark Johnson and Jukka Lehtonen, I had some inspiring and illuminating discussions.

This work has been made possible with financial support from the Department of Computer Science at Åbo Akademi University and from the Turku Centre for Computer Science. My research period at the University of Turin was financed by Stiftelsen för Åbo Akademis Forskningsinstitut and by Finnish Cultural Foundation.

Finally, I would like to thank my parents and sisters for enriching my life and my friends for reminding me that a cubicle is not a cubicle unless there is something outside of it. But the greatest thank of all goes to my family. Thank you Mavi for your love and support, and thank you Miko for showing the marvels of the world.

Turku, December 10th, 1998.

Jukka Hekanaho

Chapter 1

Introduction

The ability to gain knowledge through experience and to adapt to new situations is a profound characteristic of all natural systems. In fact, learning is the single unifying theme underlying all intelligent activities. Therefore, it is not surprising that *Machine Learning*, as the study of learning as a computational process, has always played a central role in the quest for artificial systems that exhibit intelligent behavior. In this thesis we focus on inductive inference, or *inductive concept learning*, in which learning is accomplished by examination of externally supplied examples. The task of a learner in this setup is to reason about the examples in order to learn descriptions that reveal general patterns in the examples.

Inductive concept learning has been, and continues to be, one of the most intensively studied fields within Artificial Intelligence. Its popularity stems from several factors, of which we highlight only two. Firstly, as computers are applied to more knowledge-intensive tasks, it becomes more and more laborious to extract accurate knowledge from domain experts. This knowledge acquisition bottleneck [11] is a key factor decelerating the development of expert system. A partial solution to this is to have learning algorithms which allow computers to extract knowledge from examples. Secondly, concept learning problems are ubiquitous in the real world. Thus, there is a need for good concept learners which learn to classify in complex real world domains.

In this thesis we concentrate on *supervised concept learning from preclassified examples*. This means that we know in advance a class of concept names and that we have a set of examples, each of which is preclassified to belong to a certain class. The task of a *concept learner*, also called an *inducer* or just a *learner*, is to build a classifier that can be used to classify new unseen examples. In the case of symbolic concept learners the classifier is described in symbolic terms, i.e. terms that are understandable to humans. Such a description can be used to classify new unknown examples, to do diagnosis, or it may become a part of a knowledge base, where it can be used in different kinds of inferences.

Although concept learning has received a lot of attention and there exist many good and solid concept learners, there is still a lot of work to be done and room for improvements in many aspects of concept learning. This is partly because of the great variety of existing concept learning problems, a fact that is also mirrored in the plenitude of concept learning paradigms, ranging all the way from Bayesian methods and neural networks to symbolic decision tree and rule learners.

In addition to pure supervised concept learning, where concept learners have to extract information solely from the examples, we have also studied *theory revision*. Theory revision extends concept learning by introducing background knowledge in the form of a domain theory. Thus a learner may use both the initial domain theory and the set of examples to derive a classifier.

Our emphasis is on symbolic rule-based descriptions. Given a set of classified examples and a target concept, we learn a symbolic description of the concept, expressed as a flat set of rules. Since the rules are logical formulae they have precise semantics. Rules are amenable for logical inferences and can therefore easily be a part of knowledge bases. Furthermore, a flat set of rules is very intelligible, i.e. it can be easily understood by humans. As the learning paradigm we apply methods that are closely related to *genetic algorithms*. Genetic algorithms are stochastic general purpose search algorithms. They utilize a population of solutions and try to derive better solutions through recombination operators that combine information from different solutions. In this thesis we describe two genetic algorithms that we have developed. The first one, JGA, is specifically developed for supervised concept learning. The other system, DOGMA, is a population-based system that performs theory revision.

1.1 Overview of the Thesis

Although we have looked at different aspects of concept learning and applied the methods to a variety of different domains, there are three factors that remain constant in this thesis. First of all, we use the same kind of representation language in all domains. Secondly, we utilize a population–based and stochastic search algorithm in order to find good concept descriptions. Finally, we use the same internal encoding scheme for all problems.

In Chapter 2 we describe the necessary background to our work and discuss what concept learning really is and how genetic algorithms can be used to perform it. We also discuss the differences between propositional and relational learning. Chapter 3 deals with the knowledge representation language and shows how logical formulae are internally coded by our systems. In Chapter 4 we move on to the details of JGA and describe the architecture of the system.

In Chapter 5 we discuss different criteria for choosing concepts. We also show how these criteria are turned into real-valued fitness functions which express the effectiveness of different concepts in a domain.

Chapter 6 is devoted to the empirical evaluation of JGA in propositional learning. We show learning results of JGA in several domains and compare it against some well-known concept learners.

Starting from Chapter 7 we describe theory revision in the DOGMA system. We start by discussing how theory revision differs from concept learning and how DOGMA extends JGA. We also demonstrate empirically the effectiveness of DOGMA in theory revision. Then, in Chapter 8, we describe a particular application of DOGMA and show how it can be used as a postprocessor that refines very accurate classification theories.

In Chapters 9 and 10 we move on to relational learning. First we discuss the differences between relational and propositional learning and categorize different relational learning settings. Next we indicate where DOGMA belongs in this categorization and discuss how DOGMA deviates from other relational learners. Chapter 10 describes a number of relational learning problems and empirically evaluates the effectiveness of DOGMA in these domains.

Finally, in Chapter 11, we draw some conclusions of our work and point out some potential directions for future research.

This thesis is partially based on work previously reported in the following papers.

I Jukka Hekanaho. Symbiosis in Multimodal Concept Learning. In Proceedings of the 12th International Conference of Machine Learning, pages 278-285, Tahoe City, CA. Morgan Kaufmann 1995.

II Jukka Hekanaho. A GA-based Approach to Disjunctive Concept Learning. TUCS technical report series, No 72, 1996. To appear in Machine Learning Journal.

III Anders Aspnäs, Vic Cockcroft, and Jukka Hekanaho. Use of Genetic Algorithms to Learn Ligand Recognition Concepts: Application to the GPCR Superfamily. In Proceedings of the 2nd Nordic Workshop on Genetic Algorithms, pages 131-149, Vaasa, Finland, 1996.

IV Jukka Hekanaho. Background Knowledge in GA-based Concept Learning. In Proceedings of the 13th International Conference on Machine Learning, pages 234-242, Bari, Italy. Morgan Kaufmann 1996.

V Jukka Hekanaho. GA-based Rule Enhancement in Concept Learning. In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, pages 183-186, Newport Beach, CA. AAAI Press 1997.

VI Jukka Hekanaho. DOGMA: A GA-based Relational Learner. In Proceedings of the 8th International Conference on Inductive Logic Programming, pages 205-214, Madison, Wisconsin, USA. Springer LNAI-1446, 1998.

Chapter 2

Background

In this chapter we will give the necessary background to concept learning and genetic algorithms that is needed for a proper understanding of the bulk of the thesis. We will start by defining what concept learning is and what concept learners do. We will also discuss the distinction between propositional and relational learning. As for the genetic algorithms we will give a brief overview of them, emphasizing features that are relevant from a concept learning point of view.

2.1 Concept Learning

First of all we need to distinguish between *relational* and *propositional learning*. The far more common case in contemporary Machine Learning research is propositional or attribute-based learning that deals with situations where each example is a single object and where the concept description is a proposition regarding only one object. Relational learning is applied when the examples describe several objects and when the concept we want to learn is a relation of objects. The systems we have developed are capable of both types of learning.

2.1.1 Propositional Concept Learning

In this section we will give some introductory definitions for concept learning. For clarity we concentrate first on propositional learning and postpone the description of relational learning to section 2.1.3. Let us start with a definition of attributes and instances.

DEFINITION 1 *Let* $\overline{A_i}$ *be a set of attribute values* $\{a_{i1}, a_{i2}, \ldots, a_{ik}\}$. *An* instance space $\mathcal{I}$ *is the Cartesian product of sets of attribute values,* $\mathcal{I} = \overline{A_1} \times \overline{A_2} \times \ldots \times \overline{A_n}$. *An* attribute $A_i : \mathcal{I} \rightarrow \overline{A_i}$ *is a projection function from the instance space to a set of attribute values. An* instance i *is an element of* $\mathcal{I}$ *and an* example e *an element of* $\mathcal{I} \times \mathcal{C}$*, where* $\mathcal{C}$ *is a set of class names.*

We will always restrict our attention to a given *instance space*, defined in terms of a set of attributes, where each attribute has a fixed set of possible attribute values. An *attribute* is a measurable feature of an instance. The range of attribute values may be finite or infinite and they might be nominal, structured, discrete, or continuous. We however, deal directly only with discrete and nominal attributes. Thus for example continuous values have to be discretized before they can be used by our systems. An instance is described by a vector of attribute values and an example is an instance together with its classification. The set of class names is fixed and is given in advance. Notice that although there may be more than two classes, we may always bring such a multiclass learning situation down to a dichotomous setting by considering one of the classes as a target class and the rest of the classes as a countering class. Thus a situation involving n classes can be solved as n separate binary learning situations.

A concept is a set of instances and is treated as a function $c : \mathcal{I} \rightarrow \mathcal{B}$, where $\mathcal{B} = \{tt, ff\}$ is the set of truth values. In other words, a concept assigns a truth value for each instance in the instance space. We say that a concept c *covers* an instance i if $c(i) = tt$. The set of instances covered by a concept c is exactly the instances i for which $c(i) = tt$. The *concept space* $\mathcal{H}$, which defines the set of all hypotheses of a given instance space $\mathcal{I}$, is the set of all functions $c : \mathcal{I} \rightarrow \mathcal{B}$.

Let $\mathcal{I}$ be an instance space, $\mathcal{C}$ a set of class names, and $E \subset \mathcal{I} \times \mathcal{C}$ a set of examples. The *concept learning problem for class* $c \in \mathcal{C}$ is to find a concept $h \in \mathcal{H}$ such that h is complete and consistent. A concept for class c is complete if it covers all examples of class c and consistent if it does not cover any examples of the other classes.

In practice however, we do not work directly with the semantic notion of concepts defined above, but rather we define and utilize a *representation language.* This language defines a subset of the concept space and characterizes what kinds of concepts we are looking for. In this way concept learning can be broken down into two parts: first design a suitable representation language and then search for a good concept description in that language [82]. This separation is reflected in the biases of a concept learner. Each learner utilizes several biases: a language bias that defines the hypothesis space of possible concepts and a preference criterion that defines the structure of the hypothesis space. In addition, we may also talk about the algorithmic search bias that defines how the hypothesis space is searched. Using this separation we can give a second, more practical definition of concept learning.

Definition 2 (Concept Learning)

> *Given*
>> *a language of concepts* $\mathcal{L}_{\mathcal{C}}$
>> *a target class* c *from a set of classes* $\mathcal{C}$
>> *a language of examples* $\mathcal{L}_{\mathcal{E}}$
>> *a covers relation* $\in_r$ *defining how examples are covered*
>> *a set of examples* $E \subseteq \mathcal{L}_{\mathcal{E}} \times \mathcal{C}$
>
> *find*
>> *a hypothesis* $h \in \mathcal{L}_{\mathcal{C}}$
>
> *such that*
>> $\forall\ (e,c) \in E.\ e \in_r h$*, i.e.* h *is complete*
>> $\neg\exists\ (e,nc) \in E.\ nc \neq c \land e \in_r h$*, i.e.* h *is consistent.*

Although this definition is used by many learners is still leaves room for improvement, mostly because of the requirement of completeness and consistency of the final hypothesis. In practice, due to large search spaces or inconsistencies in the examples, it is often preferable to aim at simple hypotheses that are almost complete and fairly consistent. Normally we would have a function $f_{CRIT} : \mathcal{P}^{(\mathcal{L}_{\mathcal{E}} \times \mathcal{C})} \times \mathcal{L}_{\mathcal{C}} \rightarrow \mathbb{R}$ that ranks hypotheses according to different criteria. Typically f_{CRIT} balances factors like completeness, consistency, and simplicity. Hence the task of a concept learner is really to find a hypothesis $h \in \mathcal{L}_{\mathcal{C}}$ such that $f_{CRIT}(E,h)$ is maximized. More motivation for different kinds of criteria is given in Chapter 5, where we discuss the criteria for choosing a good hypothesis.

Thus in concept learning we analyze a set of examples and use them to build a description of a concept for one or more of the concept classes. In a multiclass learning situation the goal is often to find a total *classification function* $f : \mathcal{I} \rightarrow \mathcal{C}$ that partitions the instance space into class regions. Certain concept learners, e.g. decision tree learners, form automatically classification functions. In most rule-based learners the rules for different classes may overlap, and there might be subspaces in the instance space which are not covered by any rule. Consequently, these rule learners do not automatically learn a total classification function, but have to be extended with a *rule interpreter*. A rule interpreter takes a set of rule-based concept descriptions and forms a classification function out of them. Every rule interpreter has to decide what to do with conflicting rules that cover the same instances and how to form a default rule covering the uncovered instances. In the systems developed by us we have two simple and common alternatives for rule interpretation, *rule class ordering* and *majority weighting*. In rule class ordering the rule classes are ranked so that the class with the smallest number of false positive examples comes first. An instance is classified based on the first match in this ordering. In majority weighting each rule is associated with a weight, derived from the number of examples it covers, and an instance is classified based on the sum of weights from the matching rules. Both alternatives use the same method for the default rule, choose the class with most uncovered examples.

Often the most important characteristic of the classification is its predictive accuracy. To derive a fair estimate of the accuracy we usually test the learned concept description on a separate test set of unseen examples. The problem is that many times what gives the best results on the training set of examples is not ultimately the best solution. However, normally the solution should be as consistent as possible, as complete as possible, and usually also as simple as possible. A common approach to this problem is to search for simple best solutions and to try to avoid overfitting the data. The underlying principle is that a simple description matches the "real" regularities of the training examples, while leaving out the detailed "peculiarities" of that particular sample. This principle is also known as the Occam's razor [15]. However, the original formulation given by William of Occam does not quantify simplicity, so we are in need of a more formal description of the principle. One potential candidate is the minimum description length principle [104, 105], which measures the stochastic complexity of concepts. How we apply the principle is described in Chapter 5.

In our case the hypothesis space of concepts is defined through a logical language. The language $\mathcal{L}_m$ we use is almost identical to the language utilized in another concept learning system called REGAL [46, 42]. $\mathcal{L}_m$ is also similar to the VL_1 (or in the case of relational learning VL_{21}) language of Michalski [78, 76, 77]. This is a rule–based language were each rule is a conjunction of internally disjunctive atomic formulae or selectors. Each selector is a, possibly disjunctive, statement about one attribute. The exact structure of $\mathcal{L}_m$ is defined in Chapter 3, here we will give just a brief example.

Example 1 Let $\mathcal{I} = \overline{A_1} \times \overline{A_2}$, where $\overline{A_i} = \{no, yes\}$. Assume that a concept description for class xor consists of two rules: $A_1([yes]) \wedge A_2([no]) \Rightarrow xor$ and $A_1([no]) \wedge A_2([yes]) \Rightarrow xor$. A verbalization of this description is:

IF $((A_1 = yes)$ AND $(A_2 = no))$ THEN class = xor

AND
IF $((A_1 = no)$ AND $(A_2 = yes))$ THEN class = xor.

In other words the description defines the concept of exclusive or. A set of rules, like the one above, is called a disjunctive concept description, because it is equivalent to

IF $((A_1 = yes)$ AND $(A_2 = no))$ OR
$((A_1 = no)$ AND $(A_2 = yes))$
THEN class = xor

Notice that an instance is covered by a disjunctive set of rules if it is covered by at least one of the rules. □

The usefulness of a concept description depends on its purpose. Obviously accuracy is important, but also other criteria may be used. For example, if the description has to be used and understood by humans then the simplicity and intelligibility of the description are often tantamount to accuracy. Likewise, if the concept description is to be used in a knowledge base for some kind of logical inferences then generality and perhaps also conformity with the rest of the knowledge base are among the key issues.

Learning a suitable description of a concept may be difficult due to several factors. First of all, we may simply have an insufficient number of classified examples to capture a complex concept accurately. The relationship of sample size and probable concept accuracy is studied by a subfield of machine learning, computational learning theory [4, 87]. Another factor is the size of the examples. If the examples are described by a lot of attributes then capturing the "real" concept description among many random happenstance correlations might be difficult. A related concern is the complexity of the hypothesis language that is used to describe the examples. With a more powerful language we can describe the concept more accurately, but at the same time the risk of overfitting the description to the specific training sample is greater. All these factors refer to the size of the search space and of the sample, but there are also other kinds of complications. For example, the nature of the sample of examples may have an effect on the results. The sample might be biased so that it does not reflect the real underlying distribution of the concept. Also *noise* might hinder accurate concept learning. Noise, which is either a natural part of the examples or is introduced in the process of coding the examples, leads to different changes in the examples and might make some examples indiscernible. It is customary to distinguish between two types of noise: classification noise and attribute noise. The first one refers to situations where the examples might be erroneously classified. Attribute noise, again, is used in situations where some of the attribute values of the examples might be erroneously coded.

Also statistics pursues similar goals as machine learning. Many branches of statistics aim at characterizing the behavior of the class (dependent) variable in terms of the other (independent) variables. Most of statistics is focused on analyzing continuous data, whereas in symbolic machine learning discrete and nominal data has played a pivotal role. The branch of statistics dealing with non-continuous data analysis is called *categorical data analysis*, see e.g. [1, 36] for an introduction. Also other differences exist. Statistics is normally concerned with the underlying distribution of the data, whereas machine learning is usually applied in a nonparametric fashion. However, many modern statistical techniques, like k-nearest neighbors and naive Bayes, are also nonparametric. Machine learning emphasizes the simplicity and readability of the resulting description and has developed easy-to-use and efficient tools for data analysis. Statistics, again, is more concerned with what methods can be applied in a given situation and with the relevance of the conclusion. In concurrent machine learning research there is also great interest in relational learning and in theory revision, whereas statistics has concentrated on propositional data.

Despite these differences there are many similarities between machine learning and statistics and many machine learning algorithms use, at least partially, statistically motivated decisions in the learning process. It goes without saying that both fields have a lot to learn from each other and should strive, if not for unity, at least for tighter cooperation. This effort is perhaps best manifested in the field of Knowledge Discovery in Databases that tries to incorporate machine learning, statistics, and database methodology under a unified umbrella.

2.1.2 Rule-Based Concept Learning

In rule–based concept learning the target concept is described through a classification theory, composed of a set of rules. Disjunctivity in rule-based concept learning refers to situations where one rule does not adequately describe the concept, but two or more disjunctive rules are required for the concept description. For example, in Horn clause logic a disjunctive concept description is one that requires more than one clause to represent the concept. Disjunctivity is thus related both to the concept description language and to the characteristics of the concept in question.

Rule-based learners should have a flexible strategy for handling disjunctivity. There are several good reasons for this. First of all, it is often difficult to tell in advance how many rules are needed in order to adequately describe a concept. A second reason for flexibility is that even very small adjustments in the concept description language or in the optimizing criterion f_{CRIT} may imply drastic changes in the disjunctiveness of concept descriptions. A flexible strategy for handling disjunctivity may also be useful in learning single rules. An analogous situation occurs in some genetic algorithms, see e.g. [72], which apply so-called niching methods to pursue multiple solutions in order to avoid premature convergence and to find better single solutions.

The most common way of dealing with disjunctivity in rule-based concept learning is to learn one rule at a time. In this approach, see for example [76, 77, 118], the system first learns the best rule, covering as many of the examples as possible, and then deletes the covered examples from the training set. After this the cycle is repeated with the remaining examples. This approach, however, has sometimes been criticized because it may lead to non-optimal solutions, with many small disjuncts [42]. The situation is aggravated by the problems associated with small disjuncts. They are known to be more error-prone than large disjuncts [64, 98, 121] and easily confusable with small stochastic fluctuations. Some modern rule learners tackle the problems of small disjuncts and noise through statistical complexity or confidence methods. For example, the RIPPER rule learner [18] uses a complexity measure based on the minimum description length principle to control the number of rules learned from the data. An example of the use of a confidence measure can be found in the mFOIL system [32], where sophisticated Bayesian probability estimates, like the Laplace estimate or the m-estimate, are used to control the search of rules and as stopping criteria.

Another, and a very different, alternative to rule formation is to form rules by first building a decision tree. A path from the root of a decision tree to a leaf forms a rule. Therefore a decision tree indeed learns a set of rules. Often, however, there would be numerous "path-rules" and translating the decision tree into a set of rules does not really simplify the description. Therefore, the rules should be simplified and reorganized. Many times this can be done without sacrificing the accuracy of the decision tree. An example of a system where this rule formation strategy is used is the C4.5 system [99].

Yet another approach to disjunctive rule formation is to learn the whole set of disjunctive rules directly and simultaneously. This approach is used in many genetic algorithm based concept learners, e.g. [20, 53, 65, 42, 56]. This is also the approach we have chosen.

2.1.3 Relational Concept Learning

In propositional or attribute-based learning each example consists of a single object described by a n-tuple of attribute values. The simplicity of this single feature vector setup allows the use of simple propositional languages to describe the concepts and has resulted in efficient propositional learning systems that can handle large quantities of examples and exploit the statistical properties of the data. However, two of the major limitations of most of these systems are that they are not able to use any background knowledge of the domain, nor can they learn more structured forms of concepts involving relations of several objects. In relational domains each example consists of several objects and the goal is to learn a relation of objects. In order to deal with relational domains we have to extend the simple propositional formalism and use a more structured relational formalism. Typically, relational learners use some restrictions of first order clausal logic as their concept description language.

Relational learning extends propositional learning by using a richer language both for the hypothesis and for describing the examples. The target concept in relational learning is relation of objects, and this relation is described using some relational formalism. For example, the target concept could be the relation $ancestor(X, Y)$, stating that X is an ancestor of Y. The examples (and the background theory) could then show kinship relations of different people.

But when should one use relational learning? It is needed when the examples consist of a varying number of objects and when the relations between these objects are important. Throughout the rest of this thesis we will assume that the reader knows some standard concepts of predicate calculus and logic programming, see e.g. [40, 71] for an introductory description.

Currently, the most popular approach to relational learning comes from the intersection of logic programming and machine learning, from the field of Inductive Logic Programming (ILP), see e.g. [85, 70], where the basic approach is to use a logic programming setting inductively. The background knowledge, examples, and hypotheses can in principle be any well-formed logical formulae, but in practice they tend to be expressed in some restricted clausal logic. Normal use of logic programming is for deduction, i.e. it is used to derive sound consequences from an underlying logic program. In ILP the situation is *inversed*. Here, the task is to *induce* a logic program from a set of examples, given some background knowledge.

Learning first order rules implies that one has to search very large (possibly infinite) hypothesis spaces in order to form the desired concept. Also the matching itself, i.e. checking whether a formula covers an example, is in the worst case exponential in the size of the formula. As Haussler [55] has shown, learning first order logic concepts may be an NP-complete problem, even in extremely simple cases. Restricting the space of considered hypotheses and the size of the hypotheses is therefore a central issue in relational learning. Some systems, like MIS [111], assume the existence of an oracle that answers the queries of the system. Other systems, like FOIL [97], use a greedy search for the concept and restrict the hypothesis language to function-free program clauses. In JGA and DOGMA the hypothesis space is restricted through the definition of a language template that restricts the form of possible hypothesis considered.

A different approch to the matching problem of first order logic is to avoid it by using stochastic matching [109, 45]. In this approach one does not exhaustively consider all possible matchings between variables and objects, but instead one takes a stochastic sample of the matchings. Therefore, one can not always definitely say whether a formula ϕ covers an example e. Instead the matching is probabilistic and gives a certainty bound for the cover.

We can directly use the propositional Definition 2 of inductive concept learning also for relational learning. However, we have to redefine the notions of instances and examples. In the propositional case the definition of the *covers* relation between examples and concept descriptions was straightforward. Unfortunately, in the relational case there are several possible learning settings, with different definitions of the *covers* relation and with different meanings attached to the examples. Also, whereas most propositional learners emphasize the analysis of the examples, many relational learners put a strong emphasis also on domain background knowledge. As relational learning will be applied only in Chapters 9 and 10 we will defer more detailed discussion of it until then.

2.2 Theory Revision

Pure inductive learning systems use solely the examples to learn a description of the data. However, in domains where some background knowledge related to the learning task exists, pure inductive systems have easily an inherent handicap, especially if the search space is large and the number of examples is small. Integrating background knowledge and inductive learning has been widely studied within the field of *theory revision*, e.g. [29, 92, 116, 90, 12, 103], where the purpose is to use a coarse domain theory together with training examples to produce a more accurate theory of the domain. Most theory revision systems are theory–centered and concentrate on refining the initial theory through a chain of small local changes, guided by the classified examples. Examples of such local changes include addition and removal of rules, as well as changing the individual rules through the addition or removal of conditions. Such a greedy find-and-patch approach is however susceptible to getting caught in local minima; the chain of changes may lead to a non-optimal state where no single local change improves the state.

Our approach, as implemented in DOGMA and described in Chapter 7, is to build a totally new theory using the examples together with parts of the initial theory. By this we hope to reduce the susceptibility of getting trapped in local minima.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) are stochastic and parallel search algorithms that are suitable for solving complex and ill-defined problems. They were first introduced and studied by John Holland and his students [61, 19] as problem solving as adaptation through abstraction of natural evolution. For a general introduction of GAs see e.g. [47, 75, 81]. GAs encode potential solutions to a problem as a simple *chromosome-like* structures. To solve a problem GAs evolve a *population* of chromosomes by using selection and different recombination operators to develop new solutions. A chromosome in a GA consists of a number of units, called *genes*. Each gene is responsible for a certain feature which it controls by taking different values. The values of a gene are called *alleles*. A chromosome is most often represented by a bitstring, although GAs are in no way restricted to only this representation. The chromosomes are sometimes also referred to as individuals, emphasizing their role as members of a population. The selection is biased by the *fitness* of the chromosome, so that fitter chromosomes have a higher chance of surviving and producing offspring through the recombination operators. Thus we may say that GAs model, albeit in a very abstract sense, natural evolution through genetic inheritance and Darwinian struggle for survival [75].

To solve a problem with GAs one has to find an encoding of the problem into the chromosome representation, find suitable ways of selecting good chromosomes and suitable recombination operators, and develop a fitness function that ranks the "goodness" of each chromosome. The basic operation of a GA is to initialize it with a population of chromosomes and then to build new subsequent generations of the population through the application of selection and recombination operators. This archetype of GAs is depicted in Algorithm 2.1, where the next generation of population $\mathcal{P}$ is built through a chain of intermediate populations.

```
GA(maxgen) ≡
begin
  P(0) ← Initialize;
  Evaluate P(0);
  g ← 0;
  while (g < maxgen) do
        P_S(g) ← Select P(g);
        P_R(g) ← Recombine P_S(g);
        P_M(g) ← Mutate P_R(g);
        P(g + 1) ← Evaluate P_M(g);
        g ← g + 1;
  end
end
```

Algorithm 2.1: A simple Genetic Algorithm

An encoding defines a mapping from the problem domain to a suitable chromosome representation. Finding a good encoding is clearly very problem dependent, we however use the same encoding scheme in all our problems. This encoding is defined in the next section. To distinguish between the coded chromosome and the real solution in the problem domain it is customary to denote the first one as the *genotype* of the solution and the second one as the *phenotype*.

The purpose of selection is to enforce a stochastically controlled, fitness–based exploitation on the population of chromosomes. Although there are several different kinds of selection operators, they all utilize a fitness–based ordering of chromosomes. For a comparison of different selection mechanisms see e.g. [8, 48]. Two common selection operators used in GAs are the roulette wheel selection and the tournament selection. In roulette wheel selection each chromosome in the population is given a slice of a roulette wheel, where the size of the slice is proportional to the chromosome's fitness. After this the wheel is spun and one lucky chromosome gets selected to $\mathcal{P}_S(g)$. This is repeated until we have selected enough chromosomes for a new generation. In the n-ary tournament selection a set of n chromosomes is randomly chosen from the population, without any fitness–based bias. Then, in the second phase, the chromosome with the highest fitness is chosen and is put to $\mathcal{P}_S(g)$. This is then repeated until $\mathcal{P}_S(g)$ is filled. In our algorithms we apply a modified roulette wheel selection.

After selection the newly chosen chromosomes in $\mathcal{P}_S(g)$ go through a recombination phase, which leads to some changes in them. The most common recombination operator is the one–point crossover. This simple operator takes as input two parent chromosomes and builds two offspring by selecting randomly a position in the parental chromosomes and by changing the tails of the parents. We do not use the one–point crossover in our systems, but apply instead four different types of crossovers. Mutation is another kind of operator that creates perturbation. In accordance with its biological analogue, the mutation operator is applied only sporadically and makes a random local change on a chromosome. In the case of bitstring chromosomes, mutation simply flips the value of a bit at a low probability.

After the recombination and mutation phase each chromosome in $\mathcal{P}_M(g)$ is evaluated and is assigned a fitness, based on the quality of the chromosome's phenotype.

As said above GAs utilize a fitness function that defines a measure of the "goodness" of the chromosomes. The fitness function thus defines a ranking of all chromosomes and a structure on the space of all chromosomes. The fitness function is the most varying part in our algorithms. We do not use a fixed fitness function for all application domains. Instead we have several parametrized fitness functions that we may adjust to the specific needs of each domain. This gives flexibility to our systems and makes it possible to search for different kinds of concept descriptions. This is in contrast with most symbolic concept learning systems. Most of these systems utilize one single heuristic principle in their search for concept description. While the heuristic is adjustable by small changes, it usually can not tolerate bigger revisions because the whole search method is based on the heuristic. One of the negative effects of having a flexible fitness function is that it adds a set of new parameters to the system. Therefore, for fast usage of our systems we have provided all fitness functions with default parameter values. The fitness functions we use are discussed in detail in Chapter 5.

GAs are a part of a larger family of evolutionary problem solving methods called Evolutionary Algorithms [9, 10]. This family includes several methods that use ideas from the natural evolution, e.g. Evolutionary Programming [9], Genetic Programming [68], and Evolution Strategies [9]. In Evolutionary Programming the task is to evolve finite state machines on the basis of observations. Genetic Programming is a new discipline that instead of string-like chromosomes uses variable sized tree-like structures. Actually, the structures used in [68] are LISP S-expressions. Consequently, the crossover and mutation operators used in Genetic Programming are quite different from their GA counterparts. The main difference between GAs and Evolution Strategies lies in their chromosome structures. Whereas in GAs a chromosome encodes a solution in the search space, a chromosome in Evolution Strategies encodes not only the solution but also strategical parameters for the mutation operator. Usually the chromosome encodes the probability density function for its mutation. Therefore, in Evolution Strategies mutation rate and scope is not determined globally, but each chromosome carries this information within itself. Furthermore, also the mutation parameters are subject to the adaptational changes in the population. Also the emphases of the operators are different. GAs emphasize recombination operators, using mutation only sporadically to enhance diversity and to avoid premature convergence. Evolution Strategies, on the other hand, use mutation as the main search operator.

The cycle of generations in a GA leads to more fit chromosomes, i.e. to better solutions, through a balance of two central features of GAs: *exploration* and *exploitation* of substructures in chromosomes. Exploration refers to the search of new substructures in the recombination phase, whereas exploitation manifests itself as the use of good structures in the selection phase. This improvement capability of GAs was studied by Holland in the case of chromosomes consisting of equal length strings [61] and was characterized in the *schema theorem*. In the case of bitstrings a *schema* is a string consisting of the binary symbols $\{0, 1\}$ and of a "do not care" symbol $\star$ which will match both binary symbols. Therefore, a schema represents actually all bitstrings that can be constructed from it by replacing $\star$ with a binary symbol. If the search space of all bitstrings of length m is represented as an m-dimensional hypercube, where each corner corresponds to a bitstring, then it is easy to see that a schema with n $\star$ symbols corresponds to an n-dimensional hyperplane in the m-dimensional cube. For example, the schema $1{\star}0{\star}0$ defines the set $\{10000, 10010, 11000, 11010\}$ which corresponds to a 2-dimensional plane in the 5-dimensional hypercube.

Let S be a schema of length l and with r $\star$ symbols. The *order* of S, $o(S)$, is the number of fixed positions in the schema, i.e. $o(S) = l - r$. The *defining length* $\delta(S)$ of S is the distance between the first and last fixed positions. Thus, the order measures the specificity and the defining length the compactness of information of a schema. In order to talk about the schema theorem we still need to define the fitness of a schema S in a population $\mathcal{P}$ at time t. This is defined as

$$\mu(\mathcal{P}, S, t) = \sum_{s_j \in \gamma(\mathcal{P}, S, t)} \mu(s_j) / \mid \gamma(\mathcal{P}, S, t) \mid,$$

where $\gamma(\mathcal{P}, S, t)$ is the set of all strings in $\mathcal{P}$ at time t that match the schema S and $\mu(s_j)$ is the fitness of string s_j. In other words, the fitness of a schema in a population at time t is the average fitness of all matching strings in the population at that time. Let us still denote by $\mu(\mathcal{P}, t)$ the average fitness of a population $\mathcal{P}$ at time t and by p_c and p_m the probability of crossover and mutation, respectively. Using these definitions we can give Holland's definition of the schema theorem that characterizes the exponential growth of good schemata in a GA.

THEOREM 1 [Schema Theorem]

$$\gamma(\mathcal{P}, S, t+1) \geq \gamma(\mathcal{P}, S, t) \cdot \frac{\mu(\mathcal{P}, S, t)}{\mu(\mathcal{P}, t)} \cdot [1 - p_c \cdot \frac{\delta(S)}{l-1} - o(S) \cdot p_m]$$

For a derivation and a proof of the theorem we refer to [61, 75]. Rephrased in English, the schema theorem expresses that short, low-order schemata with above average fitness receive exponentially increasing trials in subsequent generations of a GA. Thus the GA will, after initial random exploration of the search space, start to devote more and more of its search efforts to the promising parts of the search space. The short, low-order schemata can be thought of as building blocks for the final solution, i.e. the solution is achieved by combining such schemata. Therefore the success of a GA in a particular problem depends, at least partially, on the existence of such building blocks for the problem. This is expressed by a common, so far unproved hypothesis of the GAs [75].

Hypothesis 1 (Building Block Hypothesis) *A GA seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks.*

The schema theorem has been criticized because it only takes into consideration the sampling behavior of schemata and the destructive effects of mutation and crossover. Therefore, the theorem neglects the search of new chromosomes caused by the crossover and mutation operators. Furthermore, Altenberg [2] has shown that in order to get good results from GA-based search there must be a correlation between the distribution of the schema fitness and the offspring fitness. Nowadays, there exists a lot of different theoretical investigations of why the GAs work. The most popular of these is probably the application of stochastic models in the form of Markov chains, for an introduction see [102].

2.4 GA-based Concept Learning

GA–based concept learners are population–based and stochastic learners that try to solve the concept learning problem by using genetic chromosomes. Usually the chromosomes are represented as bitstrings, although also other higher level representations are used. Better classification theories are built through the search mechanism of GAs. This means that new classifiers are built from the genetic chromosomes by stochastically recombining good chromosomes. The whole process is guided by a Darwinian "survival of the fittest" principle, where the more fit chromosomes are more likely to survive and to reproduce.

2.4.1 Multimodality

Recall that a disjunctive concept learning problem requires that the learners have to induce many disjunctive rules from the examples in order to find a complete concept description. This cannot be done directly with a standard GA because the population in a standard GA tends to converge towards a single best chromosome. These kinds of problems usually require variable sized solutions and are called multimodal problems in the GA literature. A problem is *unimodal* if it can be solved with a single chromosome. Otherwise the problem is *multimodal*.

Thus in our case, multimodality is synonymous with disjunctivity and a multimodal problem is solved by a set of disjunctive rules. Hence, we will use the terms disjunctivity and multimodality as synonyms. We prefer to use multimodality when we refer to the search in GAs and reserve disjunctivity to the logical aspects of rules. The are two possible ways of solving a multimodal problem in GAs. Either you do a representation trick and code the whole multimodal problem as a single chromosome. Or, alternatively, you change the GA in order to maintain several distinct chromosomes in the population. There are also some GA-based concept learners [118, 6] that avoid the problem of multimodality by learning only one rule at a time. The first approach, usually labeled as the Pittsburgh approach, leads to variable length chromosomes and specialized recombination operators. This coding scheme has been used by various researchers in various systems, e.g. in LS-1

[112], GABIL [20], SAMUEL [53], and GIL [65]. The second approach, usually called the Michigan approach, uses fixed-length chromosomes, but has to rely on special *niching methods* to maintain diversity in the population. Examples of systems following this approach are GOGIN [51], the GA developed by McCallum and Spackman [73], the Classifier systems [62], and the early versions of REGAL [44]. With its fixed-length chromosomes the Michigan approach can use the standard genetic operators, and thus draw experience from the years of theoretical and comparative research put into these. Chromosomes in Pittsburgh type systems are more complex. This leads to more complex and non-standard genetic operators. On the other hand, the definition of the solution is easy for the Pittsburgh approach, it is enough to let the population converge to a single best solution. In the case of Michigan approach we are not as fortunate, here we have to use difficult strategies, built on top of the basic GA, in order to maintain multiple solutions in the population. At the moment, it seems that both approaches have their problems and advantages, and furthermore, it is still too early to say whether or not one approach will turn out superior. This has prompted some researchers [42, 56] to develop hybrid systems by selecting good features from both approaches and by avoiding the difficulties. Our approach is a novel combination of the two main approaches, where we keep both the fixed-length chromosomes and the simple fitness based selection of a standard GA.

JGA is a combination of Michigan and Pittsburgh approaches together with a symbiotic niching component that can be used in multimodal concept learning. On the level of fixed-length chromosomes JGA uses normal genetic operators like mutation and different crossovers. Each fixed-length string represents a single rule. In order to be able to deal with multimodal concepts the system uses a new level of operators above the basic genetic level. The operators on this level put together and divide groups of chromosomes, building families out of single, cooperating chromosomes. The families, which are composed of rules, represent complete classification theories. In this way the system keeps Michigan type effective fixed-length chromosomes and corresponding simple crossover operations. JGA also avoids the problems of multiple solutions by using Pittsburgh type families, so that the system may converge to a single highly fit family. To enhance diversity in the genetic populations JGA incorporates also a symbiotic niching component that partitions the population of chromosomes into different species. The basic idea is to identify different types of rules with different species and to gather these rules together in classification theories by using the family operators. DOGMA extends JGA to theory revision by incorporating background knowledge into the learning process.

Most GA-based concept learners are propositional. The notable relational exceptions include REGAL [44, 42], G-NET [3], SIAO1 [6], JGA, and DOGMA. JGA, DOGMA, G-NET, and REGAL use the same representational framework derived from the INDUCE system [77]. Also the chromosome representation of these GAs is the same and allows the use of standard mutation and crossover operators. SIAO1 learns first order Horn clauses by using an AQ-like covering approach. The chromosome are represented very differently from the other GAs by encoding both the predicates and their arguments as genes. This means that a mutation operator may, for example, change a predicate into a more general predicate, something that is not allowed in JGA or REGAL. However, the encoding leads to restriction on how crossover may be applied.

2.4.2 Theory of Niches and Species

In nature a fact contributing to diversity in the environment is the existence of different niches and species. An analogy of this was carried over to GAs by Holland [61] in his *Theory of niches and species.* The basic idea is to let different chromosomes occupy different niches of the environment (fitness landscape). By changing the fitness function, so that the chromosomes have to share their environmental fitnesses with other similar chromosomes, different niches can form stable subpopulations. Several different methods, all based on some measure of similarity between the chromosomes, for forming and maintaining stable niches have been proposed.

Crowding was proposed by DeJong [19] as a method for learning multimodal functions. Crowding is designed to work with overlapping populations, where the normal random replacement is changed so that similar chromosomes are more likely to replace similar chromosomes. This is done by selecting a small set of chromosomes from the old population and by replacing the most similar one with the new chromosome. In this way genetic pressure is higher amongst similar chromosomes which enhances the formation of subpopulations.

A more direct approach to the formation of niches and species is to encode the pressure for diversity directly in the fitness function, as is done in sharing functions [49, 26]. The basic idea with sharing is that chromosomes can not use the totality of their fitnesses, but have to share their fitnesses with other nearby chromosomes. Sharing may be phenotypical or genotypical, depending on whether the distance measure is defined on the parameter space or the gene space. Sharing has been used also in concept learning GAs

[73, 44, 58].

In crowding only a small fixed number of chromosomes has to be compared. Therefore, it does not add noticeable computational overhead to a GA. Unfortunately, crowding has been shown to give worse results than sharing [26]. On the other hand, sharing is computationally inefficient and requires that all chromosomes are compared to each other in order to derive the neighborhood. A more effective version of crowding, called deterministic crowding, has recently been developed by Mahfoud [72]. Mahfoud shows that this form of crowding performs at least as well as sharing on a set of test problems of varying difficulty.

Both crowding and sharing rely on a distance measure between the chromosomes. This presents some problems in case of relational learning because we are not aware of a unique general purpose distance measure in first order logic. Instead several special purpose distance measures are applied in relational concept learning, see e.g. [33, 108]. But even in propositional logic, where the definition of a distance metric between the chromosomes may be easy, the use of sharing is laborious due to the difficulties of determining suitable neighborhood ranges [58]. Fortunately, several studies [93, 113, 56] indicate that we might do without precise distance measures and use instead explicit labels to mark different subpopulations. Two chromosomes are considered similar if they have the same label. Chromosomes with different labels form different subpopulations. Spears [113] uses restricted mating and sharing together with labels and obtains roughly the same kinds of results as Goldberg and Richardson [49] get by using sharing functions with exact distance measures.

Different sharing and crowding methods lead to the coexistence of several niches in the GA population. In concept learning GA this behavior can be used to find a set of coexisting rules such that all positive examples are covered. However, this strategy may lead to insufficient global solutions. In rule-based concept learning we want to find a set of rules such that the criterion f_{CRIT} is maximized. Optimizing the equilibrium of a number of niches does not necessary lead to an optimal set of rules for f_{CRIT}. The concept learning problem seems to call for another type of approach where the chromosomes cooperate in order to form a global solution and where the utility of the global solution affects the niches.

2.4.3 Coevolution

In coevolutionary systems multiple populations evolve together. The above methods for speciation may be used to perform coevolution. Also other kinds of coevolutionary systems have been proposed. In competitive coevolution [60, 91] two species evolve by competing against each other. An alternative approach is to use multiple populations, where each population evolves a species specialized in some aspects of the global solution. In order to obtain coevolution the populations interact with each other. Two recent machine learning systems adopting this approach are G-NET [3] and the cooperative coevolutionary GAs applied in learning decision rules [94]. G-NET learns a set of rules by using a genetic level and a supervisor. The genetic level, composed of a set of genetic nodes, searches for rules covering different examples. Each node is a niche specialized in one positive example. Coevolution is achieved through a supervisor, which periodically collects the best rules from the genetic nodes and forms a global solution out of them. The supervisor also guides the genetic search by assigning different examples to different nodes. Cooperative coevolutionary GAs differ from G-NET in that there is no global supervisor node. Instead the populations cooperate by evaluating their chromosomes together with the *best* chromosomes from all other populations. In other words, the fitness of a chromosome depends on how well it can cooperate with the current global solution composed of the best chromosomes in other populations.

2.4.4 Symbiosis as Species Formation

The original theory of niches and species in GAs is based on the identification of similar chromosomes (species) and on the introduction of additional genetic pressure through competition among similar chromosomes occupying the same niche. A very different scheme is considered by us in [56, 57] and is further developed here. The scheme we apply in JGA and DOGMA is based on symbiotic cooperation of chromosomes in different species. Whereas sharing and crowding enhance competition among similar chromosomes, symbiosis promotes cooperation of dissimilar chromosomes. In symbiosis chromosomes of different species form groups and cooperate. Like its natural analogue, symbiosis in JGA and DOGMA is beneficial for the chromosomes and for the whole group if the symbionts occupy different functions in the ecology and if they, in some way, can help each other. Thus, in a way, symbiosis lifts the ecological competition on the level of symbiotic groups. Symbiosis may support the formation and maintenance of species by protecting them and by offering suitable ecological niches for them. Symbiosis differs also from the coevolving population approach in two respects. First, the cooperation in symbiosis is local whereas in the above coevolving populations the cooperation is achieved by a global scheme. Secondly, in the coevolving populations the chromosomes within a species compete directly with each other, whereas in symbiosis the competition is lifted on the level of symbiotic groups.

In JGA species are identified by explicit labels. Symbiosis is achieved through special join and break operators that put together and split apart groups of chromosomes of different species. Symbiosis, and thus also multi-modality, is maintained by lifting the fitness on the level of groups.

Chapter 3

Knowledge Representation

We use a logical knowledge representation language $\mathcal{L}_m$ to describe possible concepts. The language is basically the same as the one used in the REGAL system, although some minor differences exist. In this chapter we will describe the knowledge representation scheme used in JGA and DOGMA and show how the logical language can be mapped to a fixed-length bitstring representation.

3.1 Rule Language

We can apply our learning algorithms to different learning domains, so that the specific attributes and classes vary. However, we use always the same encoding scheme and a similar representation language for our problems. This means that although the particulars of our language may vary its structure remains always the same. The representation language is defined with a *language template* that defines the hypothesis space and how formulae are encoded as bitstrings. For clarity, we continue with the separation of propositional and relational learning started in Chapter 2 and deal first with the encoding of attribute-based domains. This separation is made only to facilitate the description, in reality both relational and propositional cases are encoded in the same manner by treating propositional learning as a special case of relational learning.

3.1.1 Propositional Language

In the propositional case each classified example consists of only one object, which is described by a list of attribute values. The language $\mathcal{L}_1$ that we use to describe propositional concepts is similar to the VL_1 language developed by Michalski [78]. The language is a rule-based language that describes a concept through a set of rules. Each rule is a conjunction of selectors, and each selector is an internally disjunctive statement about one attribute.

DEFINITION 3 *A* selector $A_k([a_{k1}, \ldots, a_{km}])$ *for an attribute* A_k *is a disjunctive formula* $A_k = a_{k1} \vee \ldots \vee A_k = a_{km}$*, where each* a_{ki} *is an attribute value for* A_k*. A* rule *for class c is a formula* $S_1 \wedge \ldots \wedge S_n \rightarrow c$*, where each* S_i *is a selector for a different attribute and c is a name of a class. A* concept description *for a class c is a set of rules for class c.*

The use of internally disjunctive selectors distinguishes our language from the representational formalisms of most other rule-based systems. The motivation for using internal disjunction is simply that it is a very compact and intelligible way of representing information. Thus it seems to offer a natural way of organizing information in a way that is easily understandable to humans.

Actually, in the language we use there is an extension to the selector definition. In addition to the type of selectors used in VL_1, we use the special symbol $*_i$ to stand for a predefined disjunction of attribute values. For each selector S where $*_S$ is used, it stands for a disjunction of all values not explicitly mentioned in the corresponding selector of the language template. This allows a more compact representation of an attribute because it is possible to collapse rare attribute values into a single symbol.

DEFINITION 4 *Let* $\overline{A_i} = \{a_{i1}, \ldots, a_{ik-1}, a_{ik}, \ldots, a_{in}\}$ *be a set of attribute values. Furthermore, let* $* = a_{ik} \vee \ldots \vee a_{in}$*.* S_i *is a* complete selector *for attribute* A_i *iff* S_i *is a disjunction of all the possible values for* A_i*. In other words,* S_i *is a complete selector if either* S_i *is* $A_i([a_{i1}, \ldots, a_{in}])$ *or* S_i *is* $A_i([a_{i1}, \ldots, a_{ik-1}, *])$*. Similarly,* S_i *is an* empty selector *for* A_i *iff* S_i *is the empty disjunction* $A_i([])$*.*

Observe that a complete selector is logically equivalent to *true* since it holds for all possible instances. The empty selector is an absurdity, it does not define any of the values of the corresponding attribute. Therefore, the empty selector is logically equivalent to *false*.

EXAMPLE 2 Assume that the set $\overline{made\_in}$ contains all the countries in the world. However, if the objects under consideration would be saunas, then only Finland and Sweden would normally be used and all other countries would be rare exceptions. If this is the case then the user might want to collapse all other countries to $*_{made\_in}$ by defining the complete selector $made\_in([Finland, Sweden, *_{made\_in}])$ in the language template. Then, all the distinguishable values of attribute $made\_in$ will be Finland, Sweden, and $*_{made\_in}$. Consequently, $*_{made\_in}$ can be used in the rules of the language, as in $made\_in([Finland, *_{made\_in}])$. This is equivalent to saying $made\_in([Finland]) \vee made\_in([*_{made\_in}])$, which can be equivalently expressed as $\neg made\_in([Sweden])$. $\square$

DEFINITION 5 *A* language template *is a conjunction of complete selectors.*

The language template defines a language $\mathcal{L}_1$ for defining rules. Each possible rule in the language is formed by deleting some parts from the language template.

EXAMPLE 3 Let $\mathcal{I} = \overline{Shape} \times \overline{Size} \times \overline{Origin}$ be an instance space, where $\overline{Shape} = \{tr, sq, ro\}$, $\overline{Size} = \{1, 2, 3, 4, 5, 6\}$, and $\overline{Origin} = \{EU, US\}$. A possible language template Λ for $\mathcal{I}$ is

$$\Lambda \stackrel{\text{def}}{=} Shape([sq, *_{Shape}]) \wedge Size([1, 2, 3, *_{Size}]),$$

where $*_{Shape} = tr \vee ro$ and $*_{Size} = 4 \vee 5 \vee 6$. A possible rule in the language defined by Λ is $Shape([sq]) \wedge Size([1, 3])$. $\square$

As can be seen from example 3, we drop the consequent part of the rules and do not include them in the language templates. We may do this because we always learn one class at a time, so the target class is always implicitly known. In our learning methodology the user always defines a language template for his problem. In this way he may choose the important values for each attribute and decide whether to collapse some attribute values together. He may also choose to neglect some attributes, or even to define new attributes through functions based on the original attributes. Observe that this specifies a VL_1-like language for the problem, where the language template is the most general formula, i.e. the language template is logically equivalent to *true*. All other formulae in the language can be obtained by deleting some parts from the template.

3.1.2 Relational Language

In the case of relational learning the knowledge representation scheme is slightly more complicated and corresponds closely to Michalski's relational

VL_{21} language [76, 77]. In this case each classified example consists of a set of objects, and the purpose is to learn a relation that characterizes the positive examples. Consequently, we must extend the selector definition to include variables. All other definitions from the propositional case can be left intact.

DEFINITION 6 *A* relational selector*, or an* internally disjunctive literal*, $A_k(X_1, \ldots, X_n, [a_{k1}, \ldots, a_{km}])$ is a disjunctive formula $A_k(X_1, \ldots, X_n, a_{k1}) \vee \ldots \vee A_k(X_1, \ldots, X_n, a_{km})$.*

EXAMPLE 4 Assume for example that we want to learn a concept h that describes a relation between two objects. Given the language template

$$\begin{aligned}\Lambda \stackrel{\text{def}}{=} & \; size(X, [small, medium, large]) \wedge shape(Y, [square, triangle, round]) \\ & \wedge \; on(X, Y, [yes, no]),\end{aligned}$$

and classified examples of concept h, for example,

$$\begin{array}{lll} \text{example1:} & + & size(a, small) \wedge position(a, 1) \wedge shape(a, square) \wedge \\ & & size(b, medium) \wedge position(b, 2) \wedge shape(b, triangle) \\ \text{example2:} & - & size(c, small) \wedge position(c, 1) \wedge shape(c, round) \wedge \\ & & size(d, medium) \wedge position(d, 2) \wedge shape(d, round) \wedge \\ & & size(e, large) \wedge position(e, 3) \wedge shape(e, round), \end{array}$$

we could learn a classification theory, expressed as a set of rules. For example, from the above examples we might suggest the following rules, which are complete and consistent with respect to the two examples above and which are expressible in the language defined by Λ:

$$\begin{array}{ll} rule1: & size(X, [small, medium]) \wedge shape(Y, [square, triangle]). \\ rule2: & size(X, [small]) \wedge shape(Y, [triangle]) \wedge on(X, Y, [yes]). \end{array}$$

□

Notice that each example is actually composed of a set of objects. Furthermore, an example only describes the properties of the individual objects. This means that in order to derive relations like $on(X, Y, [yes])$ we have to define them in terms of the original object properties. For example, $on(X, Y, [yes, no])$ is defined to be *yes* if the position of X is greater than the position of Y and *no* otherwise. Since JGA and DOGMA learn one target concept at a time we know that these rules are for the class h. Therefore, the rules actually stand for two generalized, internally disjunctive, clauses that together define the concept h.

$$\begin{array}{ll} size(X, [small \vee medium]) \wedge shape(Y, [square \vee triangle]) & \rightarrow h. \\ size(X, [small]) \wedge shape(Y, [triangle]) \wedge on(X, Y, [yes]) & \rightarrow h. \end{array}$$

The fact that relations like $on(X, Y, [yes, no])$ must be defined through the object properties in the examples may be seen as a restriction. The framework can, however, easily be extended with extensional definitions of relations.

There are two main differences between propositional and relational learning in our setting. First of all, the examples in the relational case consist of sets of objects, not of single objects as in the propositional case. The second difference is that the selectors in $\mathcal{L}_1$ are first order literals that define relations between objects, not simple attributes like in the propositional case. Thus we may say that propositional learning is a special case of relational learning where all the selectors talk about one single variable and where each example consists of a single object. The matching of a relational formula to a relational example is more complicated than in the propositional case. We say that a relational formula $\phi(X_1, \ldots X_n)$ covers an example e iff $\forall\ X_i.\ \exists\ obj_i \in e.\ \phi(obj_1, \ldots, obj_n)$ is true.

3.2 The Language Template

The full language $\mathcal{L}_1$ consists of conjunctions of selectors. The formulae in the language are mapped to bitstrings using a *language template* Λ that defines the structure and complexity of the bitstring formulae. Λ is the maximal conjunctive formula representable by the GA. The purpose of Λ is to define the hypothesis space of the GA by posing a structure on the acceptable formulae. Thus Λ acts as a language bias. All other conjunctive formulae can be obtained by deleting parts of Λ. A typical language template has the structure $\Lambda \stackrel{\text{def}}{=} \Phi(X_1, \ldots, X_n)$, where Φ is a conjunction of complete selectors. Each selector in Λ is complete, i.e. the selector contains all possible values $[v_1, \ldots, v_k]$ (possibly through the use of $*$) that the selector can take. Therefore, each selector in the language template is tautologically true, and Λ is the most general formula in the language.

As an example let us consider the language template in Figure 3.1. It defines a language for describing relations between two objects. Notice that, by definition, every selector occurring in Λ is a complete. Every possible concept description is obtained by deleting some part of Λ. For example,

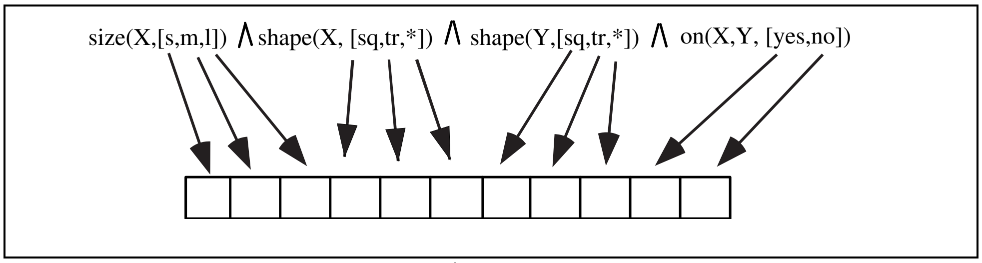

Figure 3.1: A language template

$$size(X, [s, m]) \wedge shape(Y, [tr]) \wedge on(X, Y, [yes]), \tag{3.1}$$

is a rule derived from Λ by deleting the shape selector of variable X and some internal disjuncts from the template. Formula (3.1) defines a scene with a small or medium size object that is on the top of a triangle. Notice that any literal that is a complete selector is logically equivalent to *true* and can thus be deleted from the formula without changing its extension.

3.2.1 Constructing the Bitstrings

JGA represents concepts by bitstrings. Therefore, we must map each formula in $\mathcal{L}_1$ to a bitstring. This is done by allocating one bit for each value occurring in the internal disjunctions of selectors in Λ.

The formulae are mapped to bitstring according to the following procedure. For each value belonging to a selector in Λ, set the corresponding bit to 1 if the value is present in the selector. Otherwise, if the value is not present, set the bit to 0. As an illustration, Figure 3.2 shows the mapping of formula (3.1) to a bitstring. Since Λ has a fixed length, all possible bitstrings will also have a fixed length. Observe that, because of the semantics given by the mapping procedure, a 1 in the bitstring means that the corresponding value belongs to the internal disjunction of its selector, while a 0 expresses the absence of the value. Unfortunately, this does not define a one-to-one mapping between $\mathcal{L}_1$[1] and the fixed-length bitstring, the exception being the case when the bitstring has a substring corresponding to all values of a selector set to 0's. This substring would be the result of an application of the mapping procedure to a selector with no possible values. In other words, the substring corresponds to an empty selector and is consequently

[1]We are assuming that the selectors and the values in internal disjunctions of $\mathcal{L}_1$ are always ordered as in Λ.

an absurdity. We solve this problem by rewriting all such selector substrings consisting of only zeros to substrings consisting of only ones. For example, the fourth, fifth, and sixth bit of the bitstring in Figure 3.2 will be set to 1. This means that all the values of the selector are allowed, i.e. the subformula corresponding the selector substring is logically equivalent to *true*. In other words, we go from absurdity of a selector to the most permissive statement for the selector. Notice that we could have easily retained the absurd bitstrings in the language and relied on the fitness function of the GA to punish them. However, out of efficiency reasons, we have decided to use the above rewriting scheme. It promotes the search of general formulae, which is what we most often are aiming at.

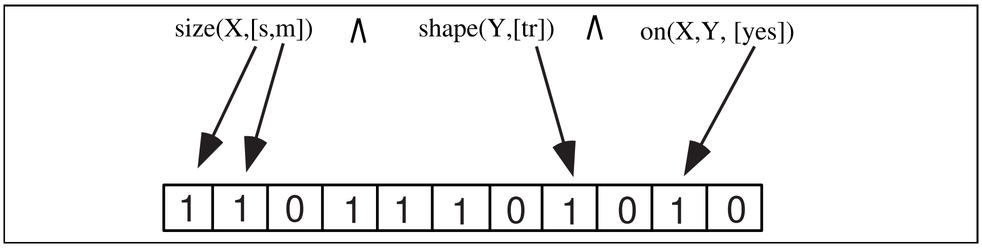

Figure 3.2: Mapping a formula to a bitstring

This renewed mapping procedure indeed defines a bijective mapping between $\mathcal{L}_1$ and the modified fixed-length bitstring. Thus we can say that bitstrings represent formulae in the language $\mathcal{L}_1$ and that JGA performs a search in the space of $\mathcal{L}_1$ formulae. Observe that the concept $true \rightarrow h$, which covers all possible examples, corresponds to a bitstring where all bit values are turned to 1. The inverse concept $false \rightarrow h$, which does not cover any examples, is not part of the language $\mathcal{L}_1$.

The mapping procedure gives very clear roles for 1's and 0's in the bitstrings. A 0 at any position is always a constraining factor, meaning that the corresponding value of the selector is not feasible. A 1 has the opposite meaning. Thus bitstrings with many 0's in them correspond to less general formulae, and increasing the 1's makes bitstrings more general. Notice that this means that turning a 0 to 1 is always a generalizing operation. The corresponding formula becomes more general since a constraining factor is removed. This may happen in two ways. Changing a 0 to 1 means either that we add a term to the internal disjunction of a selector, or that we drop the whole selector because all of its possible values are in the internal disjunction and the selector is thus complete. A selector substring consisting of only 1's corresponds to a complete selector and is always true, so the corresponding selector can be dropped from the description without changing its extension. Turning a 1 to a 0 is most often a specializing operation. In all but one situation its meaning is the opposite of turning a 0 to a 1. Normally, turning a 1 to a 0 corresponds to the removal of a term from the internal disjunction of a selector. However, if the removal of the term leads to an empty selector then the corresponding selector becomes illegal and the whole selector is logically equivalent to $false$. In this case we rewrite the whole selector by turning all the bits in the corresponding substring to 1, so that the selector becomes complete. Since a complete selector is tautologically true, this actually corresponds to dropping the whole selector from the description. Consequently, in this special case, turning a 1 to a 0 is a generalizing operation.

3.2.2 The Rule Lattice

In Figure 3.3 we show the structure of the search space we employ. Basically we are working in the complete lattice defined by all bitstrings of a fixed length. The partial order in the rule lattice $\mathcal{R}$ is defined by

$$s_i \leq s_j \stackrel{\text{def}}{=} \forall \text{ bit positions } k.\ (s_i(k) = 0 \vee s_j(k) = 1),$$

where $s_i(k)$ is the value of the kth bit of string s_i. This order is also a generality order for the corresponding $\mathcal{L}_1$-formulae, i.e. if $s_i \leq s_j$ then the formula corresponding to s_j is at least as general as the one corresponding to s_i.

Once again, we have an extra complication with the lower part of the lattice, where the bitstrings correspond to absurd formulae. Since all these bitstrings are logically equivalent to $false$, we denote this part as the *domain of falsehood* and notice that this part is actually not a part of the search space. Each absurd bitstring is mapped to another bitstring by removing the absurdity.

The examples can be translated into labeled bitstrings that are quite specific, i.e. they appear quite low in the lattice. In fact, the examples would normally lie just above the domain of falsehood. In propositional learning each example is represented by exactly one bitstring, whereas in relational learning there might be many optional bitstrings for an example, depending on how the objects are bound to variables. In either case different examples may be mapped to the same bitstring because of discrepancies introduced by the language template. Any element of the lattice that is above the domain of falsehood corresponds to a rule in the language $\mathcal{L}_1$. A rule r covers an example e if r matches e. In propositional learning this takes place iff $s_r \geq s_e$, where s_r and s_e are the bitstrings corresponding to r and

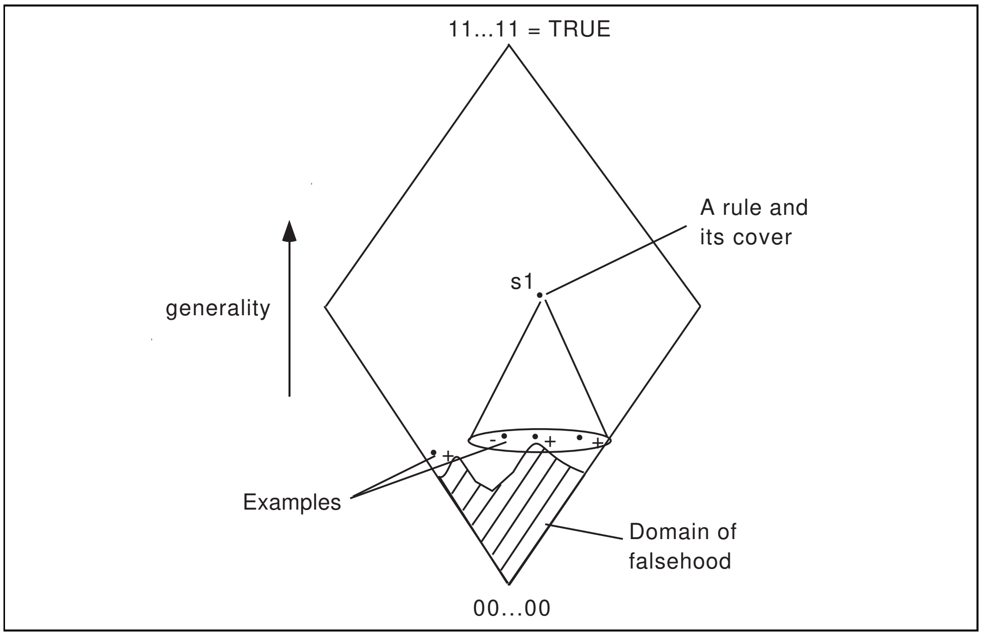

Figure 3.3: The rule lattice $\mathcal{R}$

e. In relational learning this takes place iff $\exists$ a binding b of the objects of e to the variables of the language so that $s_r \geq s_{b_e}$, where s_{b_e} is the bitstring that corresponds to e through binding b.

3.2.3 Continuous Attributes

The above coding scheme specifies what to do with discrete and nominal attributes. However, in case of continuous attributes the scheme has to be extended. Continuous attributes are dealt with by discretizing the values into fixed-length intervals after which the intervals are treated as nominal values. In this case the user defines the minimal and maximal values and the length of an interval for each continuous attribute. Given this definition JGA builds a language template by allocating one bit for each interval.

The above scheme for discretization is however rather crude and assumes that the user can give suitable interval ranges. An alternative method is to apply a discretizer. Discretizers, see e.g. [35, 67], are programs that automatically turn continuous features into intervals which may then be treated as symbolic values.

3.3 Expressing Disjunctive Concepts

The language $\mathcal{L}_1$ uses internal disjunctions, but does not include higher level disjunctions for relating different predicates. Since we are interested in learning multimodal concepts, we need to be able to express also this kind of higher level disjunctions. In order to do this we extend our language to the language $\mathcal{L}_m$ that consists of maximally m disjunctions of $\mathcal{L}_1$ formulae. At the level of bitstrings each $\phi \in \mathcal{L}_m$ is represented by a set of fixed-length bitstrings.

Therefore, in the general case the task of the concept learner is to find a set of rules for a target class. We may now reformulate Definition 2 for concept learning in Chapter 2 in terms of the rule lattice $\mathcal{R}$.

Definition 7 (Concept Learning)

Given
a language $\mathcal{L}_m$, defining a rule lattice $\mathcal{R}$
a target class $c \in \mathcal{C}$,
a language of examples $\mathcal{L}_\mathcal{E}$
a set of classified examples $E \subset \mathcal{L}_\mathcal{E} \times \mathcal{C}$,
a matcher M that checks the cover of $\Phi \subset \mathcal{R}$ on E
a criterion f_{CRIT} that evaluates the goodness of Φ
find
a set of maximally m nodes $\Psi \subset \mathcal{R}$
such that
$f_{CRIT}(M(\Psi, E), \Psi)$ is maximized.

3.3.1 Restrictions of $\mathcal{L}_m$

In addition to the internally disjunctive selectors described above, the language $\mathcal{L}_m$ may also have *necessary constraints.* These are literals without internal disjunction, like for example $square(X)$, which represent facts that must hold for every concept. The necessary constraints are used to constrain the hypothesis space and can be added to the language template as a form of background knowledge. During the inductive learning process these are only used to disallow illegal rules.

There are two other extensions to the language representation scheme: k-CNF and contiguous rules. A k-CNF rule has an upper bound k on the number of internally disjunctive terms in the selectors. For example, with 1-CNF rules there can be only one term in the internal disjunctions of a selector. Therefore, a 1-CNF rule corresponds to a Horn clause. The rule language restricted to k-CNF rules is denoted by $\mathcal{L}_m^k$. In contiguous rules each internal disjunction can only have contiguous values, where the order of the values is given by the language template. Contiguous rules are useful when applied to a predicate describing an ordered property. Such predicates result for example from discretized continuous values. Then the internally disjunctive values in the language template correspond to contiguous intervals and each internally disjunctive contiguous selector describes an interval. The contiguous language restriction is denoted by $\mathcal{L}_m^c$.

3.4 Discussion

The representational framework we use is very similar to the language that is used in the REGAL system [42]. The only differences between our language and the one used in REGAL are the k-CNF rule and contiguous restrictions of $\mathcal{L}_m$. Such language restrictions are not allowed in REGAL. $\mathcal{L}_m$ is also similar to Michalski VL_{21} language and his annotated predicate calculus language [76, 77]. However, VL_{21} uses also numerical quantifiers which are not allowed in $\mathcal{L}_m$.

In this chapter we have given a logical description of our knowledge representation framework and shown how it can be mapped into a fixed-length bitstring formalism. Unlike most rule-based systems we use a separate language for describing the examples and the hypotheses. The examples are sets of objects and the formulae are described using a logical language $\mathcal{L}_m$. In many other approaches to relational learning, most notably within the ILP school, examples are described through a subset of the hypothesis description language, as ground formulae. The advantage of this is that extension of a formula with respect to the examples can be attained through a simple logical deduction scheme, like resolution. In our setting the extension of a formula must be derived using a separate built-in matcher that bridges the cap between the example and hypothesis formalisms. However, our setting has also advantages. As described in [43], the representation we use can also easily be viewed in terms of relational algebra and object oriented databases. Furthermore, the extension of a formula can be calculated using relational algebra. This shows that the logical setting can take advantage of commercial relational database systems and that it is extendible towards large volumes of data. As such, the setting we use should also be extendible to relational data mining. In fact, Claudien, an ILP system developed by De Raedt and Dehaspe [23], uses a similar setting to perform relational data mining.

Chapter 4

The JGA system

4.1 The Architecture of JGA

JGA is a combination of Michigan and Pittsburgh approaches together with a symbiotic niching component that can be used in multimodal concept learning. The notational novelty of JGA compared to other GA-based concept learners is the concept of a family. A family is a set of chromosomes of different species. JGA learns multimodal concepts by using a two-level approach according to a metaphor of competing families. On the level of fixed-length chromosomes JGA uses normal genetic operators like mutation and different crossovers. In order to be able to deal with multimodal concepts the system imposes a new level of operators above the basic genetic level. The operators on the higher level put together and divide groups of chromosomes, building families out of single, cooperating chromosomes. To enhance diversity in the genetic populations, JGA incorporates also a symbiotic niching component that partitions chromosomes into different species.

JGA uses the same problem encoding and similar genetic operations as REGAL [42] for operating with fixed-length chromosomes. However, instead of using REGAL's strategies for learning multimodal concepts JGA implements new operators at the level of families. These operators combine and break chromosomes, introducing or canceling cooperation between them. In order to emphasize speciation JGA partitions the GA population into different species. In the speciation each chromosome is assigned to a species, and this is encoded by a natural number from the range $[0 \ldots m-1]$, where m is the number of species. This information is used for two purposes. First of all, the mating operator respects the species, i.e. the mates for crossovers are selected from the same species. Actually, the rate of crossovers between chromosomes from different species can be controlled by an interbreed parameter. Secondly, the species are taken into consideration when chromosomes are joined into families. By the definition of a family, chromosomes belonging to the same species can not be joined together. Since joined chromosomes in a family are evaluated together, this puts different genetic pressures on chromosomes of different species. The purpose is to let each species converge towards a good rule that covers some of the positive examples. At the same time the family operators are trying out and evaluating good combinations of chromosomes in different species. The families that allow more symbiosis, in the sense that the participating chromosomes together cover different examples, have a high fitness and have thus a higher chance of surviving and contributing to the next generation. The operators are implemented so that they keep the number of chromosomes in the population fixed, whereas the number of families varies from generation to generation.

JGA is implemented as a network of distributed nodes, where each node consists of a GA running on a population. The parallelization is implemented using the PVM (Parallel Virtual Machine) [39] software package. The basic nodal algorithm is given below in Algorithm 4.1. Each node works for most of the time independently from others, building new generations through the application of both chromosome and family operators. Periodically the nodes communicate with each other, sending and receiving a number of families. The families are selected for migration randomly according to their fitness. The received families replace randomly selected families in the population.

4.2 Genetic and Family Operators

In each generation JGA selects a certain percentage of the families in the population, does some operations on the selected families, and then puts the newly obtained results back to the population. As we mentioned previously, JGA has two levels of operators: a genetic level and a family level. The family operators, which work with groups of chromosomes, are *selection, join, break, make-family* and *replacement*, while the genetic operators, working with genetic chromosomes, include *mating, mutation, crossover*, and *seeding*.

$NodalJGA(maxgen)\ \equiv$
begin
  $\mathcal{P}(0) \leftarrow$ *Initialize through seeding*;
  Evaluate $\mathcal{P}(0)$;
  $g \leftarrow 0$;
  while $(g < maxgen)$ **do**
    $\mathcal{P}(g+1) \leftarrow$ *Make next generation* $\mathcal{P}(g)$;
    $g \leftarrow g+1$;
    if *epoch*;
      Send families to other populations;
      Receive families from other populations;
    end
  end
end

Algorithm 4.1: The nodal JGA algorithm

4.2.1 Genetic Operators

The mating operator mates two selected chromosomes. The operator disregards whether or not chromosomes belong to the same family. Instead mating is sensitive to the species of the respective chromosomes as specified by an interbreed parameter that defines the probability of mating chromosomes from different species. Mating in itself does not do any changes in the chromosomes, it is only used as a preparational step for the crossover operators. Mutation is a low-probability operator that randomly switches a bit value. We use four different crossovers, the well-known two-point and universal crossovers as well as a generalizing and a specializing crossover.

Given two parental bitstrings the two-point crossover produces offspring by randomly selecting two points in the parental bitstrings and by changing the bits that lie in between the two points. In this way a random sequence is changed between the bitstrings. The universal crossover operator first selects randomly a set of bit positions in the bitstrings and then produces the offspring by changing the corresponding bit values between the parents.

The last two crossover operators are specifically designed keeping in mind the specific requirements of concept learning. Recall that a bitstring is built up of substrings corresponding to each complete selector in the language template Λ. Given two parental bitstrings s_1 and s_2 the generalizing crossover works as follows. First it selects randomly a set of selectors from Λ and locates the substrings corresponding to the selected selectors in s_1

and s_2. Then, in the second step, it produces the offspring by performing a *bitwise-or* operation on the located bits of s_1 and s_2. Since a bitwise-or of two 0's always yields a 0 and of two 1's produces a 1, the only time that there is a real change is when the other string, say s_1, has a bit value 0 in a selected position and the other string, s_2, has a 1 in the same position. In this case the crossover generalizes s_1 by turning the 0 to a 1. This corresponds to the addition of a term that is used in s_2 to the internal disjunction of a selector in s_1. Since this is done for all selected positions, and the only possible change is a generalizing one, the generalizing crossover is a generalizing operator. The specializing crossover works like the generalizing one, but instead of doing a bitwise-or it performs a *bitwise-and*. Just like in the generalizing case the only real change occurs when one string, say s_1, has a 0 and the other string, s_2, a 1 in the same position. In this case the crossover turns the 1 in s_2 to a 0. This corresponds to the removal of a term from the internal disjunction of a selector in s_2, because the same term is not used in s_1. As was explained in Chapter 3, turning a 1 to a 0 is a specializing operator, except when it would lead to an empty disjunction and an illegal formula. Consequently, the specializing crossover is often, but not always, a specializing operator.

The probability at which crossover is applied to a mated pair of chromosomes is defined by the user. However, when crossover is applied the specific type of crossover is selected stochastically. We have used both a random selection of crossover type, i.e. each crossover type has a conditional probability of 0.25, and the selection scheme used by Giordana and Neri in [42]. The second selection scheme chooses the conditional probability of a crossover based on the fitness of the chromosomes and on the number of true and false positives that the chromosomes cover.

The seeding operator is similar to the seed used by Michalski in the Star methodology [77]. It generates a chromosome covering at least one randomly selected positive training example. This ensures that the initial population has useful chromosomes. The generality of the seeded chromosomes is controlled by a user-tunable generality parameter. The generality of seeding affects the balance of 0's and 1's in the initial bitstring, thus giving the user some control over the kind of chromosomes that are seeded during the run. In detail the seeding operator works as follows. First it builds a random bitstring, where each bit position is randomly generated, so that at each position there is a 1 with the probability defined by the generality of seeding. For example, using 0.1 as the value of the generality parameter leads to a bitstring where, on average, there are 10% of 1's and 90% of 0's. Next, the operator finds a binding that satisfies the randomly selected training example, after which the operator changes the bitstring minimally by turning 0's to 1's until the example is covered.

Example of Crossover Operators

In order to clarify how the crossover operators are used we will go through an example. Let Λ be the language template defined in Figure 3.1. Assume that ϕ_1 denotes the formula $shape(X, [sq, *]), shape(Y, [tr]), on(X, Y, [yes])$ and ϕ_2 the formula $size(X, [m, l]), shape(X, [sq]), shape(Y, [sq, *]),\ on(X, Y, [yes])$. Then by using Λ these formulae can be mapped into the corresponding chromosomes $i_1 = 11110101010$ and $i_2 = 01110010110$. In Table 4.1 we show the effect of some crossover operators on these two chromosomes. Operators in Table 4.1 are written as $s_{1-3,4-6}$, where s expresses the operator, in this case the specializing one, and the subscript shows which bits are affected by the crossover operator. Observe that in the case of the two-point crossover the affected bits are consecutive, and in the case of the specializing and generalizing crossovers the affected bits correspond to the substrings of the chosen literals. For example, $s_{1-3,4-6}$ corresponds to the specializing crossover being applied to the first two literals in the language template.

4.2.2 Family Operators

The remaining operators are applied on the family level. Recall that a family is, by definition, a set of symbiotic chromosomes belonging to different species. The requirement of symbiosis enforces restrictions on how families can be built. This affects the two family building operators, i.e. join and make-family. The first family operator, selection, selects randomly a multiset of families from a population. The selection is done with respect to the fitness f of the families. We use a 1-elitist roulette wheel selection that always copies the best family to the next generation.

The break operator partitions a family into two separate families. This is a simple operator that randomly chooses a number of chromosomes from a given family and builds a new family out of them.

Join is the most important family level operator, it joins two families of symbiotic chromosomes into a single family. Because of the requirements of maintaining the symbiosis, join puts only one chromosome of each species in the new family. If there are two chromosomes of the same species, one of them is deleted. The operator is made a bit more sophisticated by adding directed joining and seeding into it. When a family is about to be joined, a random uncovered positive example is chosen and join searches for a suitable

	Parents	Corresponding formulae
	11110101010	$shape(X, [sq, *]), shape(Y, [tr]), on(X, Y, [yes])$
	01110010110	$size(X, [m, l]), shape(X, [sq]), shape(Y, [sq, *]), on(X, Y, [yes])$
Xover	**Offspring**	**Corresponding formulae**
$u_{2,6,7,9}$	$1\underline{1}110\underline{01}1\underline{1}10$	$shape(X, [sq]), on(X, Y, [yes])$
	$0\underline{1}110\underline{11}1\underline{1}10$	$size(X, [m, l]), shape(X, [sq, *]), on(X, Y, [yes])$
$2p_{5-8}$	$1111\underline{0010}010$	$shape(X, [sq]), shape(Y, [sq]), on(X, Y, [yes])$
	$0111\underline{0101}110$	$size(X, [m, l]), shape(X, [sq, *]), shape(Y, [tr, *]), on(X, Y, [yes])$
$s_{1-3,4-6}$	$\underline{011100}01010$	$size(X, [m, l]), shape(X, [sq]), shape(Y, [tr]), on(X, Y, [yes])$
	$\underline{011100}10110$	$size(X, [m, l]), shape(X, [sq]), shape(Y, [sq, *]), on(X, Y, [yes])$
$g_{4-6,7-9}$	$111\underline{101111}10$	$shape(X, [sq, *]), on(X, Y, [yes])$
	$011\underline{101111}10$	$size(X, [m, l]), shape(X, [sq, *]), on(X, Y, [yes])$

Table 4.1: Examples of crossover operators. Here u and 2p stand for the universal and two-point crossovers, whereas s and g denote the specializing and generalizing crossovers. The subscripts of the operators express which bits are chosen by the operator.

family that covers this example. A family is suitable if it has a chromosome of an unused species that covers the example. If no suitable family is found then a suitable single chromosome family is constructed through seeding. After this the suitable families are joined together.

In addition to join, we have experimented with another family building operator called make-family. This is a global operator that builds a family by selecting useful chromosomes, of different species, from the population, so that the coverage of examples of the family is maximized. Basically make-family works as follows. First it selects the best chromosome from the population, marks its species as used, and remembers the examples it covers. As a second step, the operator searches the population for the chromosome of an unmarked species that best covers the remaining uncovered examples. Next, the operator marks the species of the found chromosome as used and remembers the covered examples. The make-family operator continues in this fashion while the fitness of the family increases or until a maximal number of chromosomes has been chosen to the family.

When all the operators have been applied the families are evaluated. This includes counting the cover and fitness of each family. At this point we also drop unnecessary chromosomes from each family. A chromosome is unnecessary if each example it covers is also covered by some other member in the family. Unnecessary chromosomes are dropped one at a time in a random fashion until we reach a nonredundant set of chromosomes.

Replacement builds a new generation of the population by putting back the selected families, which have now been manipulated by the various operators, into the population. Families are replaced by randomly deleting other families from the population.

The join, make-family, and break operators are implemented so that they are independent from the operators at the genetic level. Because of this we can freely mix these operators with the genetic ones. The order in which the operators are currently applied is given in Algorithm 4.2.

Make-next-generation($\mathcal{P}$) $\equiv$
begin
  $\mathcal{P}_S \leftarrow$ *Select families* $\mathcal{P}$;
  $\mathcal{P}_{S'} \leftarrow$ *Mate chromosomes* $\mathcal{P}_S$;
  $\mathcal{P}_X \leftarrow$ *Crossover* $\mathcal{P}_{S'}$;
  $\mathcal{P}_M \leftarrow$ *Mutate* $\mathcal{P}_X$;
  $\mathcal{P}_B \leftarrow$ *Break families* $\mathcal{P}_M$;
  $\mathcal{P}_J \leftarrow$ *Join families* $\mathcal{P}_B$;
  $\mathcal{P}_U \leftarrow \mathcal{P}_J \cup$ *Make-family* $\mathcal{P}$;
  $\mathcal{P}_E \leftarrow$ *Evaluate* $\mathcal{P}_U$;
  $\mathcal{P}' \leftarrow$ *Replace families* $(\mathcal{P}, \mathcal{P}_E)$;
  return $\mathcal{P}'$;
end

Algorithm 4.2: Building the next generation in JGA

4.2.3 An Example

In order to understand better how the genetic and family operators work together, we will now show an example of how one generation is turned into another in JGA. Suppose that we are working with the language defined by Λ in Figure 3.1. Assume also that we have been given a set of classified examples $E = \{e_1, \dots, e_6\}$. The examples are described in Table 4.2. Actually, the examples are given in tuple form, as described in section 3.1.2, but here we have taken the liberty of shortening the description by writing the examples in a form that is more relevant from the point of view of the language template Λ. In Tables 4.3 and 4.4 we show some useful formulae and the corresponding genotypes. Furthermore, for each chromosome, we show the species it belongs to and the examples it covers.

Name	Class	Example
e_1	+	$size(a_1,[s]), shape(a_1,[sq]), shape(b_1,[tr]), on(a_1,b_1,[yes])$
e_2	+	$size(a_2,[m]), shape(a_2,[tr]), shape(b_2,[tr]), on(a_2,b_2,[yes])$
e_3	+	$size(a_3,[l]), shape(a_3,[ro]), shape(b_3,[ro]), on(a_3,b_3,[yes])$
e_4	-	$size(a_4,[l]), shape(a_4,[tr]), shape(b_4,[sq]), on(a_4,b_4,[yes])$
e_5	-	$size(a_5,[m]), shape(a_5,[sq]), shape(b_5,[tr]), on(a_5,b_5,[no])$
e_6	-	$size(a_6,[s]), shape(a_6,[tr]), shape(b_6,[tr]), on(a_6,b_6,[yes])$

Table 4.2: Classified examples. We show the name and classification of each example.

Name	Species	String	Cover
i_1	1	11110101010	1
i_2	1	00110110110	3
i_3	2	01011001011	2, 5
i_4	2	01101001010	2
i_5	2	11011111110	1, 2, 6
i_6	1	01100100110	3

Table 4.3: Example chromosomes. In the Name and Species columns we give the names and species of the chromosomes. In the String column we show the genotypes of the chromosomes. In addition we show the cover of each chromosome.

After these preparations we are ready for a complete example. Assume that the population is made up of five families, where families $f_1 = \{i_1\}$ and $f_5 = \{i_4\}$ are single-chromosome families, and the rest of the families, i.e. $f_2 = \{i_1, i_3\}$, $f_3 = \{i_4, i_6\}$, and $f_4 = \{i_2, i_5\}$, are composed of two chromosomes. This situation is depicted in Figure 4.1, where for each chromosome we show its name, genotype, and species, and for each family we show the participating chromosomes and the cover of the whole family. Thus, the population has 8 chromosomes and five families. Suppose that the random selection selects three of the families, f_2, f_5, and f_4. Assume

Name	Formula
i_1	$shape(X, [sq, *]), shape(Y, [tr]), on(X, Y, [yes])$
i_2	$size(X, [l]), shape(X, [sq, *]), shape(Y, [sq, *]), on(X, Y, [yes])$
i_3	$size(X, [m]), shape(X, [sq, tr]), shape(Y, [tr])$
i_4	$size(X, [m, l]), shape(X, [tr]), shape(Y, [tr]), on(X, Y, [yes])$
i_5	$size(X, [s, m]), on(X, Y, [yes])$
i_6	$size(X, [m, l]), shape(X, [*]), shape(X, [*]), on(X, Y, [yes])$

Table 4.4: The phenotypes of the example chromosomes.

also that the mate operator mates two chromosomes, i_1 and i_2, in the selected families. As can be seen in Figure 4.1, the mating does not affect the selected families in any way. Next, we apply the generalizing crossover operator on the mated chromosomes. If the operator is applied to the second and third literals, as in Table 4.1, it affects bits from 4 to 9. The new offspring are $i'_1 = 11110111110$ and $i'_2 = 00110111110$. i'_1 corresponds to the formula $shape(X, [sq, *]), on(X, Y, [yes])$ that covers the examples e_1 and e_3. i'_2 again is the formula $size(X, [l]), shape(X, [sq, *]), on(X, Y, [yes])$ and covers the example e_3. The newly created offspring i'_1 and i'_2 replace their parents in their respective families. For example, family f_2 becomes f'_2, where the chromosome i_1 is replaced by its offspring i'_1. As was mentioned previously, lower-fit offspring do not always replace their parents in the families. The probability of this reproductive replacement is controlled by a survival parameter. The need for controlling the reproductive replacement can be derived from the fact that the selected units are families, and if we have large families one bad chromosome might degrade the fitness of the whole family and thus destroy the chances of many chromosomes. By controlling the replacement ratio of lower-fit offspring we can keep this degradation under control. Without such a mechanism we would have to use substantially bigger population sizes. Next in Figure 4.1, we apply two family operators. First we break the family f'_2 into two families $f_6 = \{i'_1\}$ and $f_7 = \{i_3\}$. After this we are ready to use the join operator. Suppose that the join operator is about to join family f_6. Since this family is composed of the chromosome i'_1 that covers examples e_1 and e_3 and belongs to species 1, the join operator searches for a family that has a chromosome that belongs to species 2 and covers a positive example other than e_1 or e_3. Family f_5 is such a family. Therefore, the operator may join these families together into a new family $f_8 = \{i'_1, i_4\}$. Observe that this new family is a complete and consistent solution to the learning problem in question, i.e. the classification theory composed of the two rules that correspond to the chromosomes i'_1 and i_4,

$$shape(X, [sq, *]), on(X, Y, [yes]) \rightarrow h$$
$$size(X, [m, l]), shape(X, [tr]), shape(Y, [tr]), on(X, Y, [yes]) \rightarrow h,$$

classifies all positive examples and does not include any of the negative examples. Finally, we make a new generation of the population by replacing some of the old families in the population with the current ones.

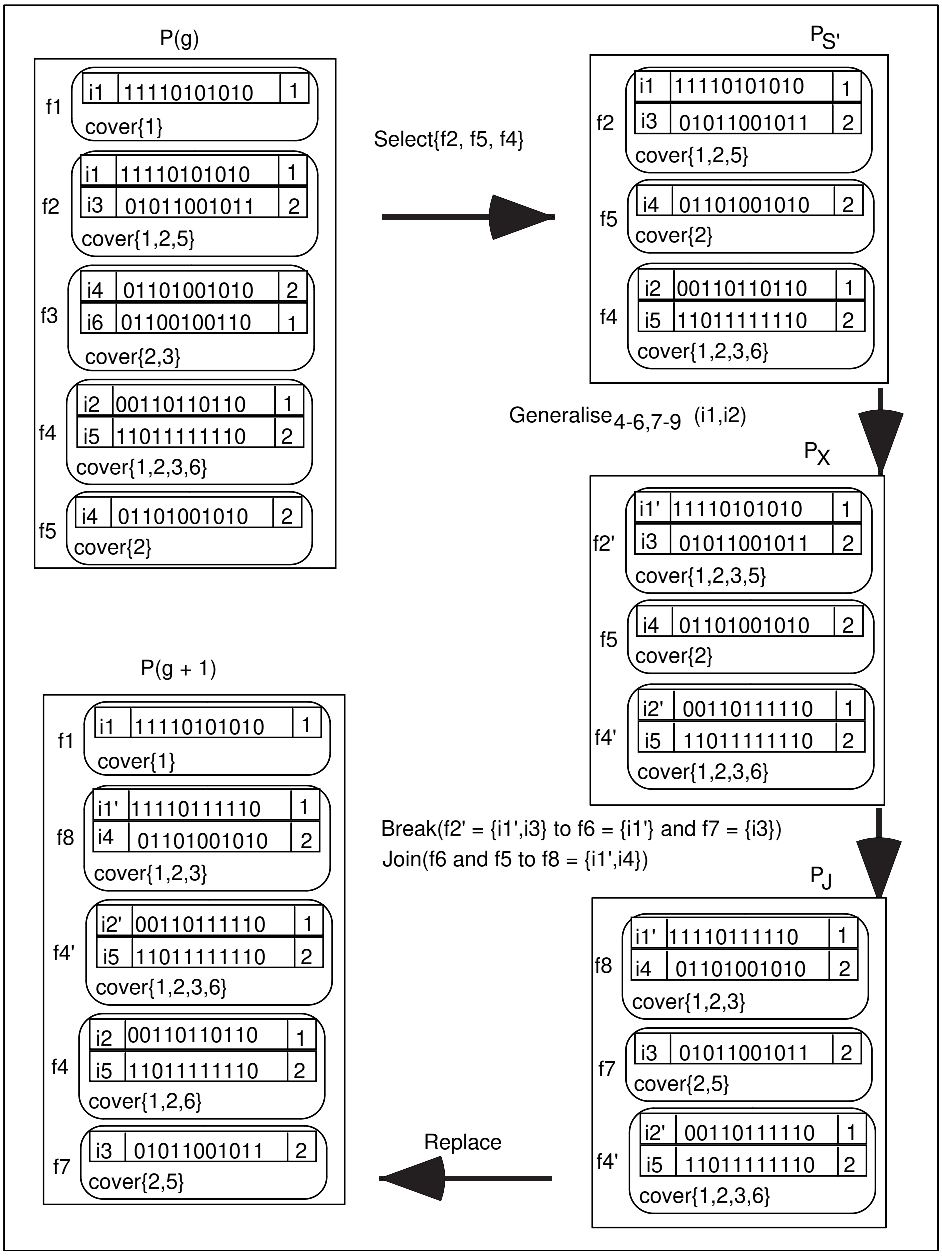

Figure 4.1: Example of how to build the next generation of a simple population with 8 chromosomes and 5 families in JGA. For each chromosome we show its name, species, and genotype. For the families we show what chromosomes they constitute of, as well as the cover of the whole family.

Chapter 5

Fitness Functions

The GA methodology emphasizes the separation of the search mechanism and the particular fitness function used to rank the chromosomes. In concept learning this separation fits in like a glove since the overwhelming diversity of concept learning problems makes it difficult, or even impossible, to construct a model selection criterion that would always give the best possible result. Whatever criterion is used to rank concepts, it adds a bias towards a class of concepts, which inevitably leads to unsatisfactory learning behavior in domains where the particular bias is unjustified [106, 50]. This is not to proclaim that there would not be general principles that could be applicable in model selection. On the contrary, we often apply ourselves a general statistical principle in model selection. Rather, we are arguing that it is sometimes difficult to balance different factors in model selection. Also from a data mining point of view we often want to investigate the available data under different presumptions.

As a result, we use three different kinds of fitness functions in JGA. The first function is a set covering fitness function, the second one is an MDL-based function, and the third function is a combination of an MDL-based measure and an information gain criterion. All these fitness functions fall into a broad class of *penalty based model selection criteria*, in the sense that they penalize the fit of the model by its complexity.

In most symbolic concept learners the measure of fitness of the description is wrapped up in the heuristic search of the algorithm. For example, many decision tree learners, e.g. [95, 99], use hill-climbing based on an information-theoretic heuristic. Such heuristics are often not amenable to major changes without changing the whole algorithm. We take instead a different approach and use several parameterizable fitness functions, which can be flexibly adjusted to different kinds of domains. We can do this because in GAs the search method is separable from the particular fitness function used.

5.1 Set Covering Fitness Function

Since we are designing a GA for concept learning from classified examples, we have to have a fitness function that rewards the covering of positive training examples and penalizes the covering of negative ones. In addition the set covering fitness uses a measure of complexity. Fitness in JGA is calculated from the whole model, i.e. from the whole family. Let E^+ be the number of positive examples in a set of examples E. Furthermore, let t_T^+ and e_T^+ be the number of true and false positives that result when we apply a classification theory T on E. The set covering fitness function f_{SC} has the following form:

$$f_{SC}(T,E) = complexity(T) \times consistency(T,E) \times completeness(T,E) + \epsilon, \tag{5.1}$$

where

$$\begin{aligned} complexity(T) &= 1 - \alpha \times \frac{\text{number of zeros in T}}{\text{length of a single bitstring}} \\ consistency(T,E) &= \beta^{-e_T^+} \\ completeness(T,E) &= (\frac{t_T^+}{E^+})^\gamma. \end{aligned}$$

Here α (≥ 0), β (≥ 1) and γ (≥ 0) are parameters that reflect how much weight we put on complexity, consistency, and completeness. ϵ is a small positive value that defines the minimum fitness. This ensures that even very unsuitable families have a possibility, even if a very small one, to be selected. The default values for the parameters are $\alpha = 0.001$, $\beta = 1.35$, $\gamma = 1.2$ and $\epsilon = 0.000001$. The default parameter settings bias the fitness function towards complete and consistent formulae. Notice that bitstrings with few 0's correspond to formulae of high generality. Thus the complexity measure promotes generality as it adds a bias towards general formulae to the system. The complexity measure will also punish formulae with unnecessarily many rules. This follows from the fact that these formulae correspond to families with many chromosomes, which will consequently have more bitstrings and thus also more 0's than more compact families. The maximum fitness is $1 + \epsilon$.

The set covering fitness function is similar to the fitness function used in the older version of the REGAL system [44]. We will demonstrate in the empirical evaluation that the set covering fitness function works well in fairly noiseless domains and if there are enough examples.

5.2 MDL-based Fitness

The second model selection criterion used in JGA and DOGMA is an MDL-based fitness function. The Minimal Description Length (MDL) principle [104, 105] and the related Minimal Message Length principle [119, 120] advocate that the best model M for describing some data D, is the one that minimizes the sum of the length of the model and the length of the data given the model. The unit of length is a binary digit. Thus to apply the MDL principle, we need to define an encoding that maps each model M_i in some model class $\mathcal{M}$ to a bitstring. Furthermore, we must also define an encoding that measures the length of the data D given a model M.

Notice that MDL does not try and find the model that maximizes the likelihood of the data, but takes also into account the complexity of the model. Therefore, the MDL principle can be view as a Bayesian MAP (maximum a posterior) estimator. Also, since in the MDL principle also the model parameters have to be encoded and minimized, the principle provides a universal yardstick for comparing models from different families. The inclusion of model complexities in concept learning is important because the inducers must often select among classification theories of different complexities. For example, in rule-based learning different rule-based theories may have a varying number of rules and the rules may use a varying number of literals.

For our purposes it is enough to consider the following simple information theoretic setup. A sender knows both the example instances I and their classification C, whereas a receiver has knowledge only of the instances I. The task of the sender is then to transmit C to the receiver, using some a priori agreed formalism. The sender may use theories T_i in some language $\mathcal{L}$ to describe and compress the classification. The total description length associated with a theory T consists of the theory cost, i.e. of the length of the encoding of T, and of the exception cost, which is the encoding of the data that is erroneously classified by T. According to the MDL principle, the best theory $T \in \mathcal{L}$ is the one that has the minimal total description length.

Our use of the MDL principle is categorical. More accurately stated, we use classification theories to determine the class of an instance and a theory always either covers or does not cover an instance. In other words, we use a 0/1-loss function. However, the MDL principle is intrinsically probabilistic, the information about an instance's class is to be expressed as the posterior probability that the instance is positive given the theory [100]. This mismatch between the approaches may sometimes, if left intact, lead to unwanted model selection behavior. To remedy the situation, we use the encoding scheme of [101] which corrects, at least partially, the displeasing categorical behavior.

As was said above, an MDL encoding defines how to encode both the theory and the exceptions in bitstrings. The exceptions are encoded exactly as in [101]. Given a theory T and a set of examples E, the exception encoding is defined as follows:

$$\begin{aligned} MDL_E(T,E) = \; & log_2(\mid E \mid + 1) + \\ & e_T^+ * log_2(\frac{e_T^+ + e_T^-}{2|E_T|}) + \\ & (\mid E_T \mid - e_T^+) * log_2(1 - \frac{e_T^+ + e_T^-}{2|E_T|}) + \\ & e_T^- * log_2(\frac{e_T^-}{|E| + 1 - |E_T|}) + \\ & (\mid E \mid + 1 - \mid E_T \mid - e_T^-) * log_2(1 - \frac{e_T^-}{|E| + 1 - |E_T|}), \end{aligned}$$

where e_T^+ and e_T^- are the number of false positives and false negatives resulting from classifying E with T, and E_T denotes the examples covered by T. This encoding penalizes an unbalance between false positives and false negatives, and performs thus slightly better than the basic encoding given in [99].

To calculate the number of bits needed to encode a rule-based classification theory $MDL_M(T)$, we encode each rule separately and then sum the bits from the individual rules. However, since we may logically send the rules in any order we need to take this into consideration, which is done by reducing the total length by $log_2(!n)$, where n is the number of rules. Each rule is a conjunction of literals. Therefore, to encode a rule we need to sum up the encoding lengths of all the literals in it, reduced by a similar logarithmic ordering measure as was used for the rules. Finally, to encode a literal in a rule, we need to look at the distribution of the examples in the possible values of the literal. Suppose that the learning setup is propositional and we want to encode $P([v_i])$, where P is an attribute in the language template with possible values $v_1, \ldots, v_n$. Then a set of examples E defines a distribution on the attribute values, based on which particular value each example has on the attribute. Let the number of examples having value v_i of the attribute P be f_i, the probability of value v_i is then $p_i = f_i / \mid E \mid$. The encoding length of attribute P becomes

$$LitBits + \sum_{v_i \in P} p_i * (log_2(\mid E \mid) - log_2(f_i)),$$

where $LitBits$ is the amount of bits needed to select among the different attributes. The sum is taken over all values of attribute P. To encode a attribute $P([v_i, \dots v_j])$ with several internally disjunctive values we use

$$LitBits + log_2(2^n - 2),$$

where n is the number of possible values of the attribute.

The above encoding works well in the propositional case. However, in relational domains it has to be extended. In relational learning each literal $P(X_1, \dots, X_m, [v_1, \dots, v_n])$ in the language template may be matched by an example $e \in E$ in several different ways, depending on how the objects in e are bound to the variables in P. Suppose there are k bindings that match the example. Then each binding adds $1/k$ to the frequency count of the corresponding literal value of the example in this binding. After this extension we may use the MDL encoding in relational learning just as we use its propositional counterpart. The final MDL encoding then combines the theory and exception encodings,

$$MDL(T, E) = W_T * MDL_M(T) + MDL_E(T, E),$$

where MDL_E is the exception encoding length, MDL_M the theory encoding length, and W_T a theory weight factor that balances the theory and exceptions. W_T is needed, since in practice the above encoding scheme underestimates length of exception bits relative to the theory bits. A possible theoretical explanation of this phenomenon is given by Quinlan in [99], where it is argued that this is caused by the redundancy present in many learning domains. Since the set of attributes used to describe the examples is often redundant, many different theories may actually cover the same set examples. W_T is used to compensate for this overlap of theories. The default value of W_T in JGA is 0.25.

To turn the MDL measure into a fitness function, we compare the MDL against the total exception length, i.e. against the description length of an empty theory $T_\emptyset$ that covers no examples.

$$f_{MDL}(T, E) = max(0, 1 - \frac{MDL(T, E)}{W_\emptyset * MDL(T_\emptyset, E)}) \tag{5.2}$$

Here $W_\emptyset \geq 1$ is a weight factor that is used to guarantee that even fairly bad theories have a positive fitness. The default value of $W_\emptyset$ is 1.2. Unfortunately however, f_{MDL} cannot be used directly as a fitness function in JGA or DOGMA. The reason is that the encoding scheme treats all exceptions, i.e. both false positives and false negatives, equally. While this is perfectly acceptable and reasonable in a comparison of final theories, it is not necessarily so while learning theories. In JGA and DOGMA the GA population consists of coexisting families which encode different theory fragments. Since the families are changed and combined in different ways, we need a fitness function that expresses not only the "absolute goodness" of a family as a theory, but also approximates the importance of the family as a building block for the final theory. Thus in our case, we need to be able to give a realistic fitness also to theory fragments. A good theory fragment is a theory that may be very incomplete but must be fairly consistent. In other words, it is a theory that may have a very high rate of false negatives but must have a relatively low false positive rate. Such a theory fragment is a good building block, in the sense that it may be merged with other fragments to form a good complete theory.

In the fitness function f_M we solve this problem by using the MDL-based fitness when we are close to complete theories, and by gradually diminishing the fitness if the theories are too incomplete or inconsistent.

$$\begin{aligned} f_M(T,E) = & \text{if } (fprate(T,E) \geq 1 \wedge tprate(T,E) \geq 1) \text{ then } f_{MDL}(T,E) \\ & \text{else if } (tprate(T,E) \geq 1) \text{ then } f_{fp}(T,E) \\ & \text{else if } (fprate(T,E) \geq 1) \text{ then } f_{tp}(T,E) \\ & \text{else} \quad f_{fp}(T,E) * f_{tp}(T,E), \end{aligned} \tag{5.3}$$

where

$$f_{fp}(T,E) = \frac{f_{MDL}(T,E) \times fprate(T,E) + conf(T) \times (1 - fprate(T,E))}{fprate(T,E)}$$

$$f_{tp}(T,E) = \frac{f_{MDL}(T,E) \times tprate(T,E) + conf(T) \times (1 - tprate(T,E))}{tprate(T,E)}$$

The MDL-based fitness is applied when rate of false positives, $fprate(T,E)$, is low enough and the rate of true positives, $tprate(T,E)$, is high enough. If the number of false positives exceeds the allowed limit, then the MDL-based fitness is reduced by a consistency measure, f_{fp}, so that the total fitness drops. Similarly, if the number of true positives is too low the MDL-based fitness is reduced by a completeness measure, f_{tp}. $conf$ is a measure of confidence of a theory fragment correctly classifying unseen cases. Currently, $conf$ is the inverse of the pessimistic error rate used by Quinlan in [99] to estimate the error rate of a single C4.5 rule or leaf. However, it could be any measure that gives a realistic fitness to a theory fragment as a building block.

The MDL-based fitness function f_M has been mostly used in DOGMA in conjunction with rule enhancement (see section 8.1). In this setup DOGMA is given a set of initial rules and its task is to use the initial rules and the examples to find a better concept description. In the case of rule enhancement DOGMA can derive the appropriate levels of $fprate$ and $tprate$ directly from the initial background rules. In other cases the user has to define the appropriate levels. In effect $fprate$ and $tprate$ are used to constrain unsuitable hypotheses, and thus guide the search to more suitable hypotheses.

Unlike f_{SC}, f_M can be used also in noisy domains where the accuracy of the learned concept description is low. We will also show in Chapter 8 that the function can be effectively used in rule enhancement. However, in many cases we do not have a set of background rules available or the background rules are so poor that we do not want to use them directly for rule enhancement. In these cases the success of f_M depends, to some degree, on our ability to define suitable levels for $tprate$ and $fprate$. To be able to efficiently work also within these kinds of domains we have defined a new kind of MDL-based fitness function. This function is the topic of the next section.

5.3 Combining MDL and Information Gain

The third fitness function we consider, f_{MG}, is easier to use than f_M and more general than f_{SC}, in the sense that it can easily be used also in noisy domains. f_{MG} is a combination of the MDL-based fitness function and of an information gain criterion.

In the previous section we touched the issue of using the MDL-based fitness f_{MDL} directly as a fitness function and mentioned that this cannot be done in JGA or DOGMA. The problem stems from the fact that f_{MDL} underrates theories that are almost consistent and very incomplete. This leads in JGA and DOGMA to a prevalence of fairly large, but very inconsistent rules since these are mostly preferred by the f_{MDL} to fairly small, but almost consistent rules. Such a dominance of inconsistent rules will, however, quite fast push the whole GA population towards the overly general and very inconsistent classification theories. The population will loose the necessary alleles for specifying the more consistent rules, and as a result the GA will spend a lot of its resources in trying to search the very inconsistent part of the hypothesis space. Thus, concept learning using f_{MDL} has *deceptive* characteristics, some schemata which are not part of the global optimum increase faster in frequency than schemata which belong to the optimum.

In the previous section the problem was solved by introducing suitable levels of false positives, and by punishing too inconsistent formulae. This, however, requires that a suitable level of false positives can be specified. Here we take a different approach and use the information gain measure.

The information gain is an information theoretic notion based on the work by Shannon in [110]. Information theory developed by Shannon is a mathematical theory of communication of messages. The word information in this context should not be confused with the semantic meaning of a message. It has more to do with what can be said than with what is said. The significant aspect is that the actual message is *selected from a set* of possible messages. Therefore, the information conveyed in a message is a measure of the degree of freedom in choosing the message. Moreover, this is measured in bits as the negative logarithm to base 2 of the probability of choosing the message.

What has this to do with concept learning? The information we are interested in when characterizing a class from a set of examples is the distribution of examples of different classes in it. Therefore, we use the definition of information and define an information measure that regards the distribution of *true and false positive examples* in a set of examples classified by a theory.

DEFINITION 8 (INFORMATION) *Let E be a set of examples, and let t_T^+ and e_T^+ denote the number of true and false positives resulting from classifying E with a classification theory T. The* information *of T in E is*

$$Info(T, E) = -log_2 \frac{t_T^+}{t_T^+ + e_T^+}.$$

The *information gain* of a theory T compared to another theory T' measures how much information is gained in the distribution of true and false positives of T compared to the distribution of T'.

DEFINITION 9 (INFORMATION GAIN) *Let E be a set of examples, and let t_T^+ denote the number of true positives resulting from classifying E with a* *classification theory* T. *The* information gain *of* T *over* T' *in* E *is*

$$Gain(T', T, E) = log_b(t_T^+ + 1) \times (Info(T', E) - Info(T, E)).$$

The definition for information gain we use is similar to the one used by Quinlan in the FOIL system [97]. The difference is that while Quinlan uses a linear factor t_T^+, we use a logarithmic factor $log_b(t_T^+ + 1)$, where $b \geq 1$. The default value of b is 1.2. We compare theories to a default theory T_{def} that classifies all examples as positives. To derive a fitness function out of the information gain we compare it against the maximal gain, i.e. against the gain of a (hypothetical) theory T_{max} that classifies all examples correctly. The maximal gain in E is $Gain_{max}(E) \stackrel{\text{def}}{=} Gain(T_{def}, T_{max}, E) = log_b(E^+ + 1) + \times (Info(T_{def}, E) - Info(T_{max}, E))$, where E^+ is the number of positive examples in E. The gain fitness is then defined as

$$f_G(T, E) = W_G \times \frac{Gain(T_{def}, T, E)}{Gain_{max}(E)},$$

where $W_G > 0$ is a parameter that can be used to adjust the gain level. Its default value is 1.2. The combined MDL and gain fitness simply selects the minimum of the two fitnesses:

$$f_{MG}(T, E) = min(f_{MDL}(T, E), f_G(T, E)). \tag{5.4}$$

In practice, f_{MG} with the default values has the effect that f_G is used in the case of specific, small rules which classify only a few examples. f_G is also used when the rules have too many false positives. In all other cases f_{MDL} is the minimum of the two functions. The effect of f_G is therefore to guide the learning in the beginning of a JGA run when the system is seeding quite specific initial rules. Another effect of f_G is to constrain f_{MG} from overrating inconsistent rules that are too general.

Chapter 6

Propositional Learning with JGA

JGA has been successfully used in several relational and propositional domains. In this chapter we will concentrate on exploring the functionality of the system in propositional domains. We will mainly use well-known datasets, taken from the repository of machine learning datasets at the University of California at Irvine [74], but we will also include some less known datasets that we have used ourselves. The empirical evaluation is two-fold; first we compare JGA with some well-established propositional learners, and then we make a more detailed comparison of JGA and C4.5 along a few selected dimensions.

6.1 General about the Evaluation

Empirical experimentation is the main method for evaluating concept learners. In most cases, the experiments are done using a collection of well-known publicly available datasets. This facilitates comparisons to other concept learners, but may also give a biased view due to the nature of the datasets. An example of a large scale comparison of a diversity of concept learners is the StatLog project [80], where 22 different statistical, symbolic and neural network based concept learners were compared in 22 different domains.

To get an unbiased estimate of the predictive accuracy of the concept learners, we perform two types of tests. The first type is the classical train and test method, where the dataset is randomly partitioned into a training and test set. The training set is used for learning a concept description, which is then evaluated on the test set. In some domains we have used the existing partition into a training and a test set. This facilitates comparison with other similar studies. In other cases, where no previous partition is made, we have used roughly 2/3 of the examples for training and 1/3 for testing. The train and test method is widely used and accepted as a valid evaluation method. For large datasets the method is expected to give fair estimate of the measure in question. However, the random nature of the partition introduces variation to the estimates, and if we have small datasets the variation can be considerable. Therefore, in order to give a more stable estimate we have used crossvalidation for some of the smaller datasets. In n-fold crossvalidation the dataset is randomly partitioned into n folds. Then the learning algorithm is executed n times, each time using a different fold for testing and the remaining $n-1$ folds for learning. We use stratified crossvalidation, where the proportions of different classes are kept roughly the same across the folds.

Irrespective of the evaluation method used, one must take care so that the training and test sets are kept separated. This means, in particular, that if some form of data transformation, like discretization, is performed, it must be done solely on the basis of the training set. Notice that if discretization is used together with crossvalidation each training fold must be discretized separately. Also the tuning of learning parameters of the systems should be done based on the training set alone. In our experiments the parameters of JGA and DOGMA were set by dividing the training set into two parts and then using one part for learning and the other part for testing the suitability of the parameters. To avoid overfitting we performed in most cases only 2-3 such initial parameter tuning runs. In the case of crossvalidation one of the training folds was used for parameter tuning. In Appendix A we show the main parameter values of JGA for all domains in this chapter.

6.2 Test Method

Since different fitness functions render JGA to perform quite differently, we will in this evaluation separately inspect two fitness functions of JGA. We reserve the name JGA_{SC} for JGA with the set covering fitness and use JGA_{MG} when the combined MDL and gain based fitness is applied. In the comparison part of the empirical evaluation we compare JGA with three well-known propositional concept learners, namely CN2 , AQ15 and C4.5.

C4.5 is a decision tree learner developed by Quinlan [99, 101]. C4.5 builds a decision tree using a delicate information-theoretic measure of the data. Included in the C4.5 package there is a tree-to-rules translator C4.5Rules that builds effective rules. The rules are generated from a decision tree in several steps, which include a simplification of the path-rules obtained from the tree and a selection of the best rule sets through an MDL-based criterion. We will compare JGA both with the decision tree and the rule versions of the C4.5 package. The AQ15 is a part of the AQ family of learning systems that have been developed by Michalski and his collaborators [79, 122]. AQ15 is a rule-based learner that is based on a set covering algorithm, the star method. CN2, again, is a rule-based learner developed by Peter Clark and Tim Niblett [17]. It combines and extends some ideas introduced in the AQ systems and in the ID3 decision tree learner.

In general, there exists a lot of different dimensions along which it is possible to investigate the performance of concept learners. Examples of such dimensions include the size of the dataset in terms of number of examples, the number of attributes and their range, the number of classes, the amount of noise, and the disjunctivity of the data. One can also gather numerous different measurements that characterize the performance of the learning algorithm. Common measurements are the predictive accuracy of the classification, the complexity of the classification, and the evaluation time of the algorithm.

We have decided to start with a general comparison of the predictive accuracies of the systems in a number of different domains of varying sizes and difficulties. We then proceed to make a more detailed analysis of the effect of the number of attributes and the number of classes on some of the systems. Finally, we will show and discuss how the error level of a domain affects the systems.

For all the evaluations we will report the accuracies of the learned classifiers and, where applicable, the standard deviations. We will also show in boldface the best accuracy achieved in each domain. Furthermore, we will underline the accuracies which are significantly worse than the best accuracy in a domain. We measure significance at the 0.05 level and use the paired t test in the 10-fold crossvalidation runs and the McNemar's test [34] in all other cases.

An important performance characteristic in GA-based systems is evaluation time. The time complexities of the runs in this chapter are reported in Appendix A. The time complexity is also investigated together with the DOGMA system in Chapter 8, where we also show some measures of the complexity of the resulting classification.

6.2.1 The Domains

We use 7 different domains in the evaluation. Most of these domains are from the machine learning repository at the University of California at Irvine (UCI repository) [74]. However, we have also included a novel dataset gathered and used only by us. The domains used are summarized in Table 6.1.

The mushrooms [107] dataset is taken from the UCI repository. It consists of 8124 examples of mushrooms classified as edible (4208) or poisonous (3916). Each mushroom is described by 22 multi-valued attributes and the problem is mapped into a bitstring of length 126. A randomly generated training set of 2000 edible and 2000 poisonous examples has been used, while the rest of the examples serve as a test set. Since the training set is quite comprehensive and noise-free, most learners do always find complete and consistent solutions.

Our second domain comes also from the UCI repository and deals with chess end-games [111, 63]. In this domain we are interested in finding rules that characterize win and no-win situations in King and Rook versus King and Pawn chess end-games, where the Pawn is one square away from queening. Each example consists of the classification together with 36 nominal attributes describing a board situation. There are totally 3196 examples classified as won (1669) and no-win (1527). In the tests reported below we use a training set of 2130 examples and leave the remaining 1066 examples as a test set. This is a domain where it is easy to achieve a very high accuracy. Consequently, JGA_{SC}, with the standard fitness parameters, should perform well.

The third problem set is the primate splice-junction gene sequences problem also from Irvine. This is a three class learning problem, where the classes correspond to intron-exon (IE) boundaries, exon-intron (EI) boundaries, and neither (N) in a DNA sequence. The dataset consists of 3190 examples which has been randomly divided into a training set of 2190 examples and a test set of 1000 examples. The examples are made up of 60 attributes, denoting 60 consecutive nucleotide positions in the DNA sequence. The task is to predict the status (EI, IE or N) of the splice position in the middle. Due to uncertainties, there is a small amount of four other attribute values than the basic nucleotides A, G, C and T. Because of this, the problem has been coded with 5 attribute values for each DNA position (A, G, C, T, *), where * stands for all the other possible values. The resulting bitstring has then 300 bits in it. We use also two other versions of the splice-junction problem for testing the effect of the number of attributes. The shorter of these versions has 12 nucleotide positions, where instead of selecting 30 nucleotide positions on both sides of the middle position, like in the original case, we chose only 6 positions on both sides. In this way we get a domain with only 12 attributes, which is mapped into a bitstring of 60 bits. The last version has 15 nucleotide positions on both sides of the middle position, giving 30 selected positions and a bitstring of 150 bits.

Another domain that is mainly used to show the effect of the number of attributes is the receptor dataset [5]. This is a dataset originally used by ourselves in collaboration with biochemists to investigate the structure of different classes of proteins, based on the ligands they bind. The dataset consists of 477 proteins classified in 8 different ligand binding categories. This is a fairly easy domain, in the sense that the classes are easily recognizable. In [5] we were mainly concerned with characteristic learning and with finding suitable k-CNF rule representations for the protein classes. What complicates the learning process is the size of the domain; there are altogether 170 attributes, each having 20 possible values. Just like the splice junction domain, this domain comes in three different versions: one having all the 170 attributes, a second having 28 attributes assumed to be near the ligand binding site and thus assumed relevant, and a third version having 70 attributes, where we have selected the 28 assumed binding site attributes plus their neighbors. The corresponding bitstring lengths of the versions are 3400, 560, and 1400. This domain also shows that JGA can quite effectively work on long bitstrings. As the test method in this domain we use 3-fold crossvalidation.

All the previous domains use only nominal attributes which is what JGA is mainly designed for. However, in order to test how JGA behaves in domains with continuous attributes, we use the domain of Australian credit screening [96], this version is from the StatLog group [80]. This domain has 14 attributes and two classes. The domain is interesting because there is a good mix of different kinds of attributes, continuous, nominal with few values, and nominal with many values. JGA can not use continuous attributes, so the only way to deal with them is to discretize them, after which JGA treats them as nominal. We have discretized the data using the entropy-based discretize utility of the $\mathcal{MLC}$++ package [66], which is based on the discretizer developed by Fayyad and Irani [35]. There are 690 classified examples in the data set. Since this is less than in most of the previous domains, we apply 10-fold crossvalidation to test the accuracies of the systems. Note that all training folds are discretized separately, and thus the bitstring lengths of different folds vary between 47 and 50 bits.

Two additional UCI domains are used in testing the effect of the number of classes. The Soybean domain [78] consists of 630 examples classified into

15 different classes. The encoded bitstrings have 119 bits in them. Although this dataset is quite small, compared to the number of classes, we use a single training and test set. The existing separation uses 290 examples in the training set and 340 in the test set. The second domain is a medical problem of lymphography [16] provided by the University Medical Center, Institute of Oncology, Ljubljana, Slovenia. The dataset consists of 148 examples divided into 4 classes. The length of the bitstring is 60 bits. To test this domain we apply a 10-fold crossvalidation.

Domain	Ex	Attr	Bits	Values	Classes	Method
Australian	690	14	$47-50$	–	2	10-CV
Chess	3196	36	73	2.0	2	TT
Lymphography	148	18	60	3.3	4	10-CV
Mushrooms	8124	22	126	5.7	2	TT
Receptor170	477	170	3400	20	8	3-CV
Receptor70	477	70	1400	20	8	3-CV
Receptor28	477	28	560	20	8	3-CV
Soybean	630	35	119	2.9	15	TT
Splice60	3190	60	300	8	3	TT
Splice30	3190	30	150	8	3	TT
Splice12	3190	30	60	8	3	TT

Table 6.1: The datasets used in the empirical evaluation in this chapter. For each dataset we report the number of examples, the number of attributes, the average number of values per attribute, and the number of classes. We also show the test method we apply in each domain.

6.3 General Comparison

In Table 6.2 we summarize the results from the general comparison. For each algorithm and each domain we report the accuracy of the method and the testing method used. For the crossvalidation runs we show also the standard deviations. In addition, we show, in boldface, the best accuracy achieved in each domain and indicate, with an underscore, which results are significantly worse than the best result.

As can be seen from Table 6.2, all systems, except C4.5Rules, learn a complete and consistent theory for the Mushrooms domain. As expected, all systems learn very accurate descriptions also in the chess domain. In this domain C4.5 and JGA have practically equal accuracy rates, whereas

Domain	Algorithm	Accuracy%	Method
Mushrooms	JGA_{SC}	**100**	TT
	JGA_{MG}	**100**	TT
	AQ	**100**	TT
	CN2	**100**	TT
	C4.5	**100**	TT
	C4.5R	99.8	TT
Chess	JGA_{SC}	99.4	TT
	JGA_{MG}	99.4	TT
	AQ	98.7	TT
	CN2	99.0	TT
	C4.5	**99.5**	TT
	C4.5R	98.9	TT
Splice60	JGA_{SC}	93.7	TT
	JGA_{MG}	**94.9**	TT
	AQ	<u>89.1</u>	TT
	CN2	<u>89.8</u>	TT
	C4.5	93.8	TT
	C4.5R	<u>92.3</u>	TT
Splice12	JGA_{SC}	94.5	TT
	JGA_{MG}	**95.7**	TT
	AQ	<u>93.2</u>	TT
	CN2	<u>92.2</u>	TT
	C4.5	<u>94.0</u>	TT
	C4.5R	<u>92.9</u>	TT
Australian	JGA_{SC}	<u>82.0</u> ± 3.6	10-CV
	JGA_{MG}	84.9 ± 3.7	10-CV
	AQ	<u>82.9</u> ± 4.9	10-CV
	CN2	<u>83.6</u> ± 6.1	10-CV
	C4.5	**87.0**±3.1	10-CV
	C4.5R	<u>85.2</u> ± 3.7	10-CV
continuous	C4.5	85.5 ± 2.3	10-CV
	C4.5R	<u>84.2</u> ± 3.8	10-CV

Table 6.2: Comparing different symbolic concept learners in 5 domains. We show the accuracies of each system. In addition, we show in boldface the best accuracy in each domain and mark with an underscore the accuracies that are significantly worse than the best one.

C4.5Rules, AQ, and CN2 learn slightly less accurate descriptions. There are, however, no significant differences between the learners in the Mushrooms or Chess domains. In the Splice60 domain JGA_{MG} achieves the highest accuracy. JGA_{SC} and C4.5 learn descriptions that are slightly less accurate, whereas the rest of the learners are significantly worse. In the smaller Splice12 domain the ranking of the systems is similar to the splice60 domain, except that AQ now achieves a higher accuracy than CN2 and C4.5Rules. The highest accuracy is achieved by JGA_{MG} followed by JGA_{SC}. All the other learners are significantly worse than JGA_{MG}. Comparing the results from Splice12 and Splice60 domains, we see that both versions of JGA and C4.5 cope well with the added dimensionality of splice60, whereas AQ and CN2 deteriorate when the number of attributes and the information available increase.

Considering Australian credit screening, we notice that C4.5 performs very well and also C4.5Rules and JGA_{MG} do learn fairly accurate rules. The rest of the learners cannot cope so well with the increased error rate of this domain. In the case of JGA_{SC} and AQ this is not surprising since the model selection criteria of these systems is geared towards complete and consistent theories. Therefore, they easily overfit in error-prone domains. The poor performance of CN2 is more surprising since it includes noise handling capabilities.

Since C4.5 is able to learn directly from continuous data, we learned descriptions with C4.5 also on the original, undiscretized version of Australian data. As the results reveal, the discretization does not deteriorate the data but is actually helpful in this case, i.e. C4.5 learns more accurate descriptions using the discretized data. This indicates that the entropy based discretization is indeed an effective way of dealing with continuous attributes, at least in this domain. For a general comparison of the effect of discretization see [30, 67].

Overall, based on the results summarized in Table 6.2, we may say that JGA learns descriptions whose accuracy is well comparable to those achieved by the other well established learners. In fact, JGA_{MG} is the best learner in many domains and among the top three algorithms in each of the tested domains.

6.4 Detailed Comparison

Having now set up a general frame of the performance of JGA with respect to other systems, we can move to a more detailed analysis along three different dimensions. To do so we will analyze the performance of JGA and C4.5 in a few domains with varying number of attributes and classes and with different error levels.

6.4.1 Number of Attributes

One of the main difficulties in concept learning is that the attributes that are important for the classification are often overshadowed by many other less relevant attributes. To investigate how JGA handles the dimensionality problem of numerous attributes we perform tests in two domains and use a varying number of attributes. The chosen domains are the splice junction domain, where we use versions with 60, 30, and 12 attribute, and the receptor domain, where we have selected 170, 70, and 28 attributes. The results are summarized in Table 6.3.

Domain	Attr	JGA$_{SC}$	JGA$_{MG}$	C4.5	C4.5Rules
Splice60	60	93.7	**94.9**	93.8	<u>92.3</u>
Splice30	30	94.3	**94.6**	93.7	93.1
Splice12	12	94.5	**95.7**	<u>94.0</u>	<u>92.9</u>
Receptor170	170	97.1 $\pm$ 0.3	**97.9**$\pm$0.8	<u>94.6</u> $\pm$ 2.1	<u>95.4</u> $\pm$ 2.3
Receptor70	70	**98.7**$\pm$0.5	**98.7**$\pm$0.5	<u>95.0</u> $\pm$ 2.4	96.9 $\pm$ 1.0
Receptor28	28	**99.6**$\pm$0.6	<u>97.9</u> $\pm$ 1.7	<u>96.9</u> $\pm$ 0.9	<u>97.3</u> $\pm$ 0.8

Table 6.3: Detailed comparison of the effect of the number of attributes. We show the number of attributes in each domain and the predictive accuracy percentages of JGA and C4.5. As the test method we used train and test in the splice domains and a 3-fold crossvalidation in the receptor domains.

As can be seen from Table 6.3 both versions of C4.5 are quite robust when the number of attributes is increased. This indicates that the greedy search, guided by an information theoretic gain criterion and aided by post pruning, is performing fairly well even when there are numerous attributes. However, in the receptor domains there is a slight degradation of the accuracy, 2.3% units for C4.5 and 1.9% units for C4.5Rules, when we move from the 28 attribute version to the 170 version. Also JGA responds robustly to an increase of the attributes. In the receptor domains JGA$_{MG}$ is the most balanced learner. JGA$_{SC}$ achieves the highest accuracy in the receptor domains. This is not surprising since the accuracy rates in these domains are very high, which agrees with the fitness bias of JGA$_{SC}$. In the splice domains C4.5 is the smoothest learner with respect to the number of attributes.

Overall, throughout the experiments recorded in Table 6.3, JGA achieves in general a better accuracy than the C4.5. In average the difference between JGA_{SC} and C4.5 (C4.5Rules) is 1.7% (1.6%) units. The average difference between JGA_{MG} and C4.5 (C4.5Rules) is 2.0% (1.9%) units. Although the accuracy tends to drop with added attributes, we may say that in the receptor and splice domains both JGA and C4.5 are able to cope well with increasing dimensions.

6.4.2 Number of Classes

JGA learns rules for one class at a time and then uses a rule interpreter to make a collective classification of all the classes. Some other concept learners, e.g. decision tree learners, learn all the classes simultaneously. In order to demonstrate that the one class at a time policy is not a serious restriction, we compare JGA against C4.5 in three different domains with various number of classes. The domains that were chosen are: Lymphography, a small medical domain with 148 examples and four classes; Receptor, the 8-class domain already used for the attribute number testing; and Soybean, a 15-class domain with 630 examples.

Domain	Classes	JGA_{SC}	JGA_{MG}	C4.5	C4.5Rules
Lympho	4	81.1 ± 9.2	**82.4**± 6.3	79.8 ± 8.4	77.7 ± 7.4
Receptor170	8	97.1 ± 0.3	**97.9**± 0.8	$\underline{94.6} \pm 2.1$	$\underline{95.4} \pm 2.3$
Soybean	15	$\underline{85.0}$	**93.5**	90.6	92.6

Table 6.4: Experiments with different number of classes. We report the predictive accuracy percentages of C4.5 and JGA. The method of evaluation was train and test for Soybean, 3-fold crossvalidation for Receptor, and 10-fold crossvalidation in the Lymphography domain.

The results summarized in Table 6.4 show that JGA is able to handle multiclass learning problems. In fact, JGA_{MG} is more accurate than C4.5 in all three tested domains. JGA_{SC} learns comparatively accurate rules in the Lymphography and Receptor domains, but is the worst learner in the 15-class Soybean domain. However, since JGA_{MG} learns an accurate description in the Soybean domain, the poor performance of JGA_{SC} is probably due to the fitness function it utilizes.

6.4.3 Error Level

To show the effect of the error rate level on the learning performance of JGA and C4.5, we selected five domains used in the previous experiments. The results presented in Table 6.5 are ordered according to the average error rate achieved in each domain. Comparing JGA$_{SC}$ with JGA$_{MG}$, we notice that when it is possible to learn almost error free rules both systems perform equally well. But as it becomes more difficult to learn accurate rules, the error rates for rules learned by JGA$_{SC}$ increase faster. This indicates that the set covering fitness function f_{SC} is biased towards complete and consistent rules and has difficulties in coping with domains where this strategy leads to overfitting. A comparison of JGA$_{MG}$, C4.5, and C4.5Rules is more difficult and nothing conclusive can be deduced from the results in Table 6.5.

Domain	JGA$_{SC}$	JGA$_{MG}$	C4.5	C4.5Rules
Chess	99.4	99.4	**99.5**	98.9
Splice60	93.7	**94.9**	93.8	<u>92.3</u>
Soybean	<u>85.0</u>	**93.5**	90.6	92.6
Australian	<u>82.0</u> ± 3.6	84.9 ± 3.7	**87.0**±3.1	<u>85.2</u> ± 3.7
Lymphography	81.1 ± 9.2	**82.4**±6.3	79.8 ± 8.4	77.7 ± 7.4

Table 6.5: Experiments on the effect of error rate level. The domains are ranked according to the average error rates. We report the predictive accuracy percentages of C4.5 and JGA.

6.5 Discussion

We have compared the JGA system, a population-based and stochastic method for concept learning, with three well-known symbolic concept learners, namely CN2, AQ15, and C4.5. In the first part of the comparison, we established that JGA achieves a comparatively high rate of predictive accuracy. JGA$_{MG}$ and the C4.5 versions seem to perform well in the first general comparison. In the second, more detailed, part we compared JGA and C4.5 along different dimensions. Overall, the empirical evaluation demonstrates that JGA$_{MG}$ is in general more robust than JGA$_{SC}$. Furthermore, the experiments indicate that JGA$_{MG}$ is highly competitive when compared to other symbolic learners. However, we should be cautious when interpreting these results and remember that JGA was given a distinct advantage over the other learners in the form of parameter tuning. It is probable that also the other learners could have benefited from initial parameter tuning. Nevertheless, the empirical evaluation indicates that JGA_{MG}, with the selected parameter tuning approach, is a robust learner and achieves a high accuracy rate.

Also some other GA-based concept learners have been applied in some of the domains of this chapter. For example, both REGAL [89] and G-NET [3] has been applied in the mushrooms and splice junction domains with good results. In the mushrooms domain REGAL and G-NET can always find simple solutions which are both complete and consistent. Comparison in the splice domain is more difficult since the REGAL and G-NET report classwise error rates, whereas we report the total error rate. Because of this we compare in Table 6.6 the classwise error rates of JGA_{MG}, REGAL, and G-NET in the splice junction domain. We show also the number of rules for each learner. REGAL and G-NET were trained with 2000 examples and tested on 1190 examples. The corresponding train and test split for JGA was 2190 and 1000.

	Accuracy			Rules		
Class	JGA_{MG}	REGAL	G-NET	JGA_{MG}	REGAL	G-NET
EI	96.5	95.6	96.6	3	19	7
IE	94.4	95.8	97.1	2	26	10
Neither	96.6	94.8	96.7	5	21	11

Table 6.6: Comparing JGA_{MG}, REGAL, and G-NET in the splice junction domain. We show the accuracy rates for each class and the number of rules for each learner. The data for REGAL and G-NET is from [3].

Comparing the results we notice that, at least in the splice domain, JGA seems to biased towards more general rules, i.e. JGA learns classification theories with less rules than REGAL or G-NET. Since both JGA_{MG} and G-NET use an MDL-based fitness function, this might indicate that the two systems encode and emphasize the theory and exception bits differently. For the EI and Neither classes JGA's fitness bias seems to be suitable; the system uses fewer rules than G-NET to learn descriptions which are as accurate as the ones learned by G-NET. However, for the IE class the fitness bias of JGA is probably too general since in this case both REGAL and G-NET are more accurate. G-NET has also been applied to the Australian credit screening domain, where it achieved an accuracy of $84.2\% \pm 4.4$. The test method was 10-fold crossvalidation. Comparing with Table 6.2, G-NET achives a better accuracy than $JGA_{SC}(82.0\% \pm 3.6)$, but is slightly less accurate than

$JGA_{MG}(84.9\% \pm 3.7)$.

An important property of GAs is the clear separation of the search mechanism and the heuristics involved in the fitness function. Because of this GAs could be used to test the effect of different learning heuristics that are used to compare different concepts. However, as we remarked in Chapter 5, in order to encode such heuristics as fitness functions in a GA one might have to refine them so that they become more suitable for the GA-based search.

We use parameterizable fitness functions that can be adapted to different domains and to different user requirements. This adds flexibility to the system, but puts also an additional burden on the user. He must now find suitable parameters for the fitness function. Therefore, JGA is not as user friendly as the other learners used in the comparison. Fortunately, there are some heuristics for selecting suitable fitness parameters.

Chapter 7

DOGMA: GA-based Theory Revision

7.1 Introduction

GA-based concept learning systems, like [51, 20, 42, 65], have been shown to be an alternative to more traditional symbolic concept learners in cases where there is little or no domain theory available. There are also some studies, e.g. [52, 65], showing how the search in GAs can be constrained and guided through additional knowledge about the search space, but overall there has been little work done in integrating background knowledge into concept learning GAs. However, in domains where there exists some background knowledge, related to the learning task, pure inductive systems have easily an inherent handicap, especially if the search space is large and the number of examples is small. In this chapter we will recapture some of our work in [57] and study how to partially remove this handicap by integrating initial domain theories with a concept learning GA-based system. Our emphasis is not in constructive induction or in theory–centered theory revision, but rather in helping and in speeding up the inductive learning process. We use the background knowledge as a basin of suggestions, or as a pool of alleles that is used to give hints to the GA of possible search directions. This leads to an approach that is more theory–suggested than theory–centered.

GAs are recognized as being able to effectively sort out promising subspaces in large search spaces. In concept learning this means that GAs may quickly point out promising substructures. It is also widely accepted that GAs are not very efficient in fine-tuning the results. In concept learning the real problem is often not that of finding a solution, but *selecting* a good one among a vast variety of alternatives. The problem is aggravated in domains where the search space is large and the number of examples is insufficient to guide the search past local optima. The problem can be made easier through background knowledge that gives partial solutions and identifies substructures that can be used in the final solution. In GAs such background knowledge could be used to guide the search by pointing out promising search directions and to speed up the process by concentrating faster on suitable hyperplanes. Of course, using background knowledge too promptly or having misleading knowledge can make the learning process more difficult by leading the search astray. In this chapter we will show how to integrate background knowledge, given in the form of a flat clausal classification theory, with a concept learning GA.

The developed system, DOGMA (Domain Oriented Genetic MAchine), is built on top of the system JGA described in Chapter 4. DOGMA incorporates background knowledge into the learning process and extends JGA to theory revision. There are two ways in which the domain theory may influence the learning process in DOGMA. The first way is through seeding; the clauses in the domain theory are turned into chromosomes through the process of background seeding. The second way in which the domain theory may affect learning is through speciation of the chromosomes based on the background context; each genetic chromosome may choose a part of the domain theory as its background knowledge, giving effectively a background context for each chromosome. This context is then used as a basis for the speciation of the chromosomes.

7.2 Background Knowledge in Concept Learning

By *background knowledge* we refer to any set of clauses, whereas a *domain theory* is the set of clauses that defines the concept classes of a learning problem. In this way background knowledge may include definitions that are generally true in the learning domain or that define beneficial constructors of the initial features, while domain theory is always strictly related to the concept classes.

The explanation-based learning framework, set forward by [27, 83], was popular in the late 1980's as an approach for using background knowledge in learning. The approach, however, suffers from the insistence of using complete and consistent knowledge. This cannot, in general, be guaranteed in real world domains.

Our purpose is to use an initial domain theory together with training examples to produce a more accurate theory of the domain. Thus our approach falls within the field of *theory revision*, see e.g. [29, 92, 116, 90, 12, 103]. Most theory revision systems are theory–centered and concentrate on refining the initial theory through a chain of small local changes, guided by the classified examples. Examples of such local changes include adding and removing rules, as well as changing the individual rules by adding or removing conditions. The focus is in repair. The theory is repaired by patching up parts of it, so that they better match the examples. These systems work mostly within the same representation, i.e. the initial classification theory and all subsequent refinements of it, including the final result, are expressed using the same representation. Some systems, however, use also a flexible representation and can change their representation as part of their refinement process. For example, the system KBANN [116] refines the theory in a totally different environment. In KBANN the initial theory is expressed as a logical formula, this formula is, however, coded into a neural network, where the theory is then refined.

Related concerns are also in focus in *constructive induction*, see e.g. [77, 31], which aims at building a theory of the domain by constructing new, more appropriate, features from the original features. However, whereas theory revision concentrates on refining the theory, constructive induction focuses on the orthogonal issue of representation. Systems performing constructive induction like MIRO [31], strive to find a compact description of the target concept by using the domain knowledge and the initial features to build up more appropriate higher level features of the training examples.

Also ILP addresses the issue of background knowledge in concept learning. However, the main focus in most ILP systems is to generalize an existing theory, mostly assumed to be correct, by adding clauses to it. In theory revision, on the other hand, the focus is in modifying an incorrect theory. In some cases ILP systems do perform theory revision, but require user input to correct faults in the theory [111, 22]. An example of a full-fledged relational theory revision system is FORTE [103]. It performs theory revision by using a hill-climbing search that is guided by a global heuristic. The operators, which make local changes to the theory, are derived from different sources, including ILP and propositional theory revision. A relational system that is similar in spirit to our approach is FOCL [92], an inductive system that integrates explanation-based learning and pure inductive learning. FOCL extends FOIL [97] by considering prior knowledge during the learning process. However, FOCL does a hill-climbing search based on an information-based metric, whereas we rely on the search mechanism of GAs.

Whereas traditional theory revision concentrates on the theory, we emphasize the learning process. We do not focus on the refinement of the theory, but rather we build a completely new theory, possibly using substructures from the background knowledge. Also, we do not do any form of constructive induction, the final classification theory we learn is a flat set of rules expressed using the basic features. Instead, we use the initial background knowledge as a basin of suggestions that may be used to guide the learning process. Like KBANN we change the representation of the initial formulae, in our case we code them into bitstrings. However, unlike in KBANN, our representation has a clear and simple semantics, allowing us to translate the final theory back to an equivalent logical formula. It is possible to extract symbolic rules also from KBANN [115], but the extracted theory is only an approximation of the theory encoded in KBANN. In GAs relevant substructures in the background knowledge can be taken into consideration by filtering them through some mechanism and then introducing the result as chromosomes in the population. This approach provides a very flexible structure because the final classification theory is not restricted by the initial domain theory. On the other hand, since we do not focus on refining the initial theory, but build a completely new one, we might actually corrupt a good initial theory. The flexibility of the structure allows the use of different kinds of background clauses, including good initial rules and approximately correct initial rules, as well as partial substructures that are not directly related to the initial concept definition. The driving force of our approach is the stochastic nature of the filtering process combined with the genetic operators and the selection pressure of the GA. This process identifies good building blocks in the background knowledge and enables them to survive, leading to faster convergence of the population.

7.3 Background Knowledge Representation

The background knowledge is given as a flat set of clauses, where the clauses are built up from literals with internal disjunctions. More formally, the background clauses are expressed by rules of the following kind,

$$\begin{aligned} cl : h \leftarrow \quad & P_1(X_{1_1}, \ldots, X_{1_m}, [v_{1_i}, \ldots, v_{1_j}]), \\ & \vdots, \\ & P_n(X_{n_1}, \ldots, X_{n_m}, [v_{n_i}, \ldots, v_{n_j}])., \end{aligned}$$

where $P_k(X_{k_1}, \ldots, X_{k_m}, [v_{k_i}, \ldots, v_{k_j}])$ are literals with internal disjunction and cl indicates the concept class in which the rule may be used.

When a background theory comprised of clauses is given to DOGMA it filters the clauses through the language template Λ and produces *background chromosomes*. The filtering procedure simply removes the head of the clauses, as well as all literals that are not a part of the language template. Then the procedure turns the remaining literals to the same form as the corresponding literals in Λ. After this, each clause corresponds to a formula in the language defined by Λ, and the formula can therefore be mapped into a bitstring using the basic mapping procedure. In this way each background chromosome corresponds to a selected clause in the background theory restricted to the language defined by Λ.

Consider, for example, the propositional setting and let Λ denote the language template

$$color([r, b, w, g]) \wedge size([s, m, l]) \wedge shape([tr, ro, *]).$$

Further, let the background knowledge be comprised of the following clauses:

$$\begin{array}{ll} cl1: & c1 \leftarrow color([r, g]), size([m]), shape([tr, sq]). \\ cl1: & c1 \leftarrow color([r, w]), size([m]), position([1, 2]). \\ cl1: & p \leftarrow color([r, b]), size([s]). \end{array}$$

Then the filtering procedure will turn these clauses into corresponding background chromosomes for class $cl1$ by filtering them through Λ:

$$\begin{array}{l} color([r, g]), size([m]), shape([tr, *]). \\ color([r, w]), size([m]). \\ color([r, b]), size([s]). \end{array}$$

The resulting background chromosomes can then be mapped into bitstrings by applying the mapping procedure. Relational theories are handled in the same manner as propositional ones.

Observe that DOGMA deals only with flat background theories. It has no reasoning mechanism that could perform logical inferences on the theories. Therefore, hierarchical theories should be flattened before they can be used as background theories in DOGMA.

7.4 DOGMA versus JGA

DOGMA is built on top of the multimodal concept learner JGA [56]. Just like JGA, the system is a combination of Michigan and Pittsburgh approaches together with a symbiotic niching component that can be used in multimodal concept learning. The system follows the metaphor of competing families by using Michigan type effective genetic operators that work on fixed-length chromosomes and by lifting selection and replacement on the level of Pittsburgh type families, which are randomly built from chromosomes to be good overall solutions. Also fitness is lifted to the level of families. The operators on the family level put together and divide groups of chromosomes. In this way the operators build families out of single, cooperating chromosomes. DOGMA extends and modifies JGA to deal with background knowledge. The main mechanism in DOGMA for incorporating background knowledge into the learning process is background seeding. Background seeding creates new chromosomes and puts them into the population. This is done by randomly selecting parts from the background chromosomes and by turning these parts into genetic chromosomes that are then put to the population.

7.4.1 Speciation

In addition to background seeding DOGMA deviates from JGA in its approach to speciation. JGA partitions chromosomes into different species by using explicit labels, which are simply natural numbers. DOGMA, on the other hand, gives the species a semantic meaning by using the specific context of the background knowledge that a chromosome has as the basis for speciation. In other words, chromosomes are partitioned into species according to what parts of the background knowledge they may use. When a chromosome is created, either in the initialization of the population or during the process of joining families, the chromosome is designated to belong to a certain species. A species corresponds to a single background chromosome, i.e. each genetic chromosome may choose at most one background chromosome as its background context. The speciation of chromosomes is used in three ways. First of all, the restricted mating operator respects the species, i.e. the mates are selected from the same species. Actually, there is an interbreed parameter determining the rate at which the species are respected in mating, so it is possible to interbreed chromosomes from different species. Secondly, the speciation affects the GA through background seeding; chromosomes may use background seeding only from the background chromosome they have chosen. Finally, speciation is also taken into consideration when the chromosomes are joined into families. Chromosomes of the same species can not be joined together. Since joined chromosomes in a family are evaluated together, this puts different genetic pressures on chromosomes of different species. The chromosomes that do not select any background chromosomes and, consequently, have not got any background context are treated in a special manner. These chromosomes are considered free, in the sense that background seeding cannot be applied to them and they may be freely joined into families. However, restricted mating continues to apply to the free chromosomes.

7.4.2 Differences on the Operator Level

Recall from Chapter 4 that the normal seeding operator used in JGA creates a bitstring that is guaranteed to cover at least one selected positive example. In addition to normal seeding DOGMA can also use background seeding. Instead of creating a random bitstring like in normal seeding, background seeding generates the initial bitstring by randomly selecting bits from a background chromosome. The bitstring is then changed minimally, so that it covers a randomly selected positive example. The amount of bits that are selected from the background chromosome is controlled by a user-tunable background rate parameter. The background seeding operator is also used while creating the initial population of DOGMA. This ensures that the GA can start off with some meaningful chromosomes.

The join operator that joins two families of symbiotic chromosomes into a single family is also slightly different than in JGA. The symbiotic requirements of join are not extended to chromosomes that have no background context. Recall that the species of a chromosome is determined by its background context, i.e. by the background rules that the chromosome may seed from. This means that join builds families with chromosomes having different background contexts. The chromosomes that have no background context are treated as free chromosomes, and the join operator may put many such chromosomes into a family.

7.5 Empirical Evaluation of DOGMA

In order to facilitate the evaluation of DOGMA, we concentrate in this section on just one fitness function, the set covering fitness function f_{SC}. The empirical evaluation is carried out by confronting three versions of the system DOGMA. DOGMA$_{null}$ is a pure inductive learner that uses only the classified examples and no background knowledge to form a classification theory. DOGMA$_{seed}$ adds background seeding to the learning process. This version uses the background knowledge by converting parts of it to genetic chromosomes, which are then put into the population. DOGMA$_{seed}$ does not, however, use background speciation. The third version is DOGMA$_{full}$, which uses the full capacity of DOGMA, i.e. it uses both background seeding and background speciation.

In order to verify the functionality of the system, we carry out an empirical evaluation in two different domains, both taken from the UCI repository. The first domain consists of recognizing promoters from a sequence of nucleotides [54]. A promoter sequence in the DNA is the region that indicates the beginning of a gene. The promoter domain has 106 examples, which are equally divided into positive (53) and negative (53) examples. Each example consists of 57 consecutive nucleotide positions of the DNA, numbered from p-50 to p7 and excluding p0, where each position can labeled by a, c, g, or t. Using all features and all possible values the problem can be mapped into bitstrings consisting of 228 bits ($4 * 57$). In this domain we want to evaluate how a domain theory affects the quality of the final result and the speed of learning. For this aim we use DOGMA with the initial domain theory in Figure 7.1. The domain theory we use is also found in the original dataset as well as in [29].

Notice that before we can use the domain theory we have to write it into the right form and flatten it. The flattened and pruned theory consists of 52 clauses. This domain theory classifies *none* of the examples. Nevertheless, the theory is still useful because it recognizes some important regions in the sequence.

We use three randomly generated training sets of 80 examples and use for each set the remaining 26 examples as a test set. We conducted the experiments according to the following procedure. After some initial experiments we fixed the parameters of the GA, e.g. the population size 100, the generation gap 0.7, the number of generations 50, and the important background rate parameter 0.98. The last parameter controls the amount of bits that are copied from the background chromosomes. Then, for all methods, we performed three experiments on each of the three training sets. As a comparison, we performed also experiments with bigger populations (200 chromosomes) and longer evaluation times (100 generations) for the DOGMA_{null} version. The results from 9 independent runs (three runs for each training set) for each version are shown in Table 7.1, where the Version column gives the version used. The Population and MaxGen columns express the population size and the number of generations used in the experiments. The SolGen column shows the the generation when, in average, a complete and consistent solution was first found, NA in the SolGen column means that in some experiments a solution was never found. Finally, the Error column shows the average error percentage of the results on the test set.

```
promoter <- contact, conformation.
contact <- min35, min10.

min35 <- p-37([c]),p-36([t]),p-35([t]),p-34([g]),p-33([a]),
     p-32([c]).
min35 <- p-36([t]),p-35([t]),p-34([g]),p-32([c]),p-31([a]).
min35 <- p-36([t]),p-35([t]),p-34([g]),p-33([a]),p-32([c]),
     p-31([a]).
min35 <- p-36([t]),p-35([t]),p-34([g]),p-33([a]),p-32([c]).

min10 <- p-14([t]),p-13([a]),p-12([t]),p-11([a]),p-10([a]),
     p-9([t]).
min10 <- p-13([t]),p-12([a]),p-10([a]),p-8([t]).
min10 <- p-13([t]),p-12([a]),p-11([t]),p-10([a]),p-9([a]),
     p-8([t]).
min10 <- p-12([t]),p-11([a]),p-7([t]).

conformation <- p-47([c]),p-46([a]),p-45([a]),p-43([t]),p-42([t]),
     p-40([a]),p-39([c]),p-22([g]),p-18([t]),p-16([c]),p-8([g]),
     p-7([c]),p-6([g]),p-5([c]),p-4([c]),p-2([c]),p-1([c]).
conformation <- p-45([a]),p-44([a]),p-41([a]).
conformation <- p-49([a]),p-44([t]),p-27([t]),p-22([a]),p-18([t]),
     p-16([t]),p-15([g]),p-1([a]).
conformation <- p-45([a]),p-41([a]),p-28([t]),p-27([t]),p-23([t]),
     p-21([a]),p-20([a]),p-17([t]),p-15([t]),p-4([t]).
```

Figure 7.1: Initial domain theory for promoters.

Version	Population	MaxGen	SolGen	Error
null	100	50	NA	35.9%
null	200	100	31	25.6%
seed	100	50	NA	26.9%
full	100	50	20	15.4%

Table 7.1: Results from the promoter domain. We show the version of DOGMA used, the population size, and the number of generations. Furthermore, we show the generation when a complete and consistent solution was found and the error rate of the solution. The results are averages from three training sets with three independent runs for each set.

```
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-4([A]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-4([G]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-4([C]).
ei1 <- p-3([C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-4([T]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-5([A]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-5([G]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-5([C]).
ei1 <- p-3([A,C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-5([T]), p-4([T]).
ei1 <- p-3([C]), p-2([A]), p-1([G]), p1([G]), p2([T]), p3([A,G]),
    p4([A]), p5([G]), p6([T]), p-5([T]).
```

Figure 7.2: The domain theory EI1 for donors in the splice-junction domain.

The second domain we use is the splice-junction problem [114] already used in the evaluation of JGA, see section 6.2.1. Here we will concentrate on learning donors, i.e. the EI class. In this domain we will use two different domain theories, EI1 and EI2, which are defined in Figures 7.2 and 7.3.

The first theory, EI1, is the splice junction domain theory in [114]. It identifies some useful structures and correctly classifies 4% of the examples in the EI class. The second theory, EI2, is a very general classification theory that correctly classifies about 96.5% of the exon-intron boundaries, but it also misclassifies about 3% of the other classes. There are totally 3190 examples in the dataset. We used two different training sets, both made of 200 randomly chosen examples, and used for each set the rest of examples (2990) as a test set. On both training sets we performed four experiments for each method. Like in the promoter domain the GA parameters were fixed after some initial tests, this time we used a population size of 100, a generation gap of 0.7, and 50 as the number of generations. The background rate parameter was set to 0.9 for EI1 and to 0.8 for EI2. For comparison we also performed some heavier tests (population size 1000 and number of generation 100) for the case with no background knowledge. The results are summarized in Table 7.2, where the Version and Theory columns show which version of DOGMA and which domain theory was used. The Pop-

```
ei2 <- p1([G,C]), p2([T]), p4([A]), p5([G,T]), p6([G,T]).
ei2 <- p-6([G,C]), p1([G]), p2([T]), p4([A]), p6([T,C]).
ei2 <- p-1([A,G,C]), p1([G]), p2([T]), p3([A]), p5([G,T,C]),
    p6([A,T,C]).
ei2 <- p1([G]), p2([T]), p5([G]), p6([T]).
ei2 <- p-6([A,G,T]), p-5([G]), p-1([G,C]), p1([G]), p2([T]),
    p5([G]).
ei2 <- p-6([A]), p-1([G,C]), p1([G]), p2([T]), p5([A,G]).
ei2 <- p-6([A,G,C]), p-5([A,G,C]), p-1([G,C]), p1([G]), p2([T,C]),
    p3([G,T]), p5([G]).
ei2 <- p-6([C]), p-4([C]), p-3([A,G,C]), p-1([A,G,T]), p1([G]),
    p2([T,C,*]), p4([A]).
ei2 <- p-5([A]), p-3([A,C]), p1([G]), p2([T,C]), p3([A,T,C]),
    p5([G,T]).
ei2 <- p-1([A,G,C]), p1([G]), p2([T,C]), p3([A]), p4([A,G]),
    p5([A,G,T]).
ei2 <- p-4([A,G,T]), p1([G]), p2([T,C]), p3([A]), p5([G]).
ei2 <- p-6([A,T]), p-3([A,G]), p1([G]), p2([T]), p5([G]).
ei2 <- p-6([A,G,T]), p-5([A,T,C]), p-1([A,G]), p1([G]), p2([T,C]),
    p3([A,G]), p4([A,G,C]), p5([G,C]),p6([A,G,T]).
```

Figure 7.3: The domain theory EI2 for donors in the splice-junction domain.

ulation and MaxGen columns show the population size and the number of generations used in the experiments. The SolGen column gives information about the generation when, in average, a complete and consistent solution is first found, NA in the SolGen column means that in some experiments a solution was never found. Finally, the Error column shows the average error percentage of the results on the test set.

Although one should be careful in drawing general conclusions from tests in only two domains, the experiments and the results shown in Tables 7.1 and 7.2 indicate that the null version, i.e. DOGMA$_{null}$, that uses a normal population size has clearly the worst behavior. Even when the null version is run with bigger populations and for a longer time it still leads in most cases to a worse performance than the seed and full versions. In the promoter domain the full version, DOGMA$_{full}$, is clearly the best method and also in splice-junction it seems to have a slight edge over the seed version. Overall, however, it is still too early to say whether speciation has an advantage over pure background seeding.

In Chapter 6 we saw that the MDL based fitness function f_{MG} gives a slightly better accuracy in the splice-junction domain. Thus, we could probably improve the above results for donors by using another fitness func-

Version	Theory	Population	MaxGen	SolGen	Error
null	-	100	50	NA	28.2%
null	-	1000	100	25	7.4%
seed	EI1	100	50	NA	6.9%
full	EI1	100	50	28	5.7%
seed	EI2	100	50	22	6.5%
full	EI2	100	50	8	6.5%

Table 7.2: Results in the splice junction domain for class EI. We show the version of DOGMA used, the domain theory, the population size, and the number of generations. Furthermore, we show the generation when a complete and consistent solution was found and the error rate of the solution. The results are averages over two training sets with four independent runs for each set.

tion. This however, does not change the overall picture — DOGMA can take advantages of useful background knowledge.

7.6 Conclusions

In this chapter we have discussed an approach to integrate background knowledge with concept learning GAs. We have presented a GA-based system called DOGMA that learns a rule-based concept description using both background knowledge, in the form of a flat set of clauses, and a set of examples. Our aim has not been to perform constructive induction, nor do we emphasize theory–centered theory revision. Our goal is to use existing background knowledge in the process of learning by directing the search of the GA. Because of the stochastic nature in which we apply background knowledge and due to the flexibility of the GA, we may use different kinds of background knowledge, including correct or incorrect domain theories and definitions of constructors that may be useful in the learning process. We have tested the use of background knowledge both via background seeding and through speciation. The results demonstrate that background knowledge can be used in GAs to save computational efforts and to get better learning results.

Chapter 8

MDL-based Rule Enhancement

8.1 Introduction of Rule Enhancement

GAs have been used both in propositional and relational concept learning, see e.g. [6, 20, 42, 53, 56]. GAs are, in general, successful in avoiding local minima and produce often near optimal solutions, provided that they are given enough resources. However, the heavy computational burden of GAs is sometimes mentioned as a problem of these systems. In the other extreme of computational complexity we have fast and more or less greedy concept learners, see e.g. [17, 18, 99]. These systems learn fairly good, sometimes even very good, classification theories with only a fraction of the time required by GAs. However, a fast greedy learner might get trapped in local minima that might sometimes be quite far from the solution that is globally most feasible. An ideal algorithm would first learn a fast approximation of the data and then with more time and resource available would produce better and better approximations.

A partial solution to the problem was offered by us in [57] and was further developed in [59]. In these articles we describe the GA-based theory revision system DOGMA. The system takes as input preclassified data and a rule-based classification theory of the data. From these system tries to build a new, enhanced theory that more accurately models the given data. Although DOGMA was originally developed for revising quite incomplete and inaccurate background theories, it may also be applied to fairly accurate theories. This offers a partial solution to the time complexity problem of GAs since we may now use a fast concept learner as a preprocessor, to produce a first approximation of the data, and then take the produced theory and refine it in DOGMA in order to gain a hopefully better second approximation.

In this chapter we describe an application of DOGMA to MDL-based theory revision in supervised concept learning. To do so we first produce a classification theory using a rule-based concept learner. Then, in the second step, we use the produced theory as an initial background theory in DOGMA. We use the MDL-based fitness function f_M (see Chapter 5). This fitness function is suitable for rule enhancement because of two reasons. Firstly, since the function is MDL-based it balances complexity and coverage of the rules. Secondly, the fitness function keeps track of suitable levels of false positives and false negatives and reduces the fitness of rules not satisfying these conditions. The suitable levels are automatically generated from the initial background rules. Thus the GA can disregard large portions of the search space and move fast towards rules that satisfy the coverage conditions.

We have selected C4.5Rules [99] as the initial rule learner. There are several reasons as to why C4.5Rules was selected. First of all, C4.5 is a well-known and quite robust learning system that produces a good set of rules in most domains. Hence, there is a real challenge in refining the rules. Secondly, C4.5 uses the MDL principle to select the final rule set. Since we are using DOGMA with the MDL-based fitness function f_M, where the MDL encoding is the same as in C4.5, we may compare the two systems also on the basis of the final MDL encoding of the theories.

8.2 Empirical Evaluation

In order to evaluate the ability of DOGMA to enhance classification rules, we conduct a set of experiments using the following simple setup. First, run C4.5Rules on the problem and translate the rules into DOGMA background rules. Then, as a second step, run DOGMA on the same problem using the translated rules as an initial theory. We use this setup in 8 well-known concept learning domains, all taken from the UCI repository [74]. The datasets and the test method used in the evaluation are described in Table 8.1. In each domain we measure the predictive accuracy, number of rules, and the description lengths of both the initial and final classification theories. We measure also the response time T of DOGMA, defined as $T = max(T_{ij})$, where T_{ij} is the evaluation time of process j for concept class i. The time is expressed in CPU-minutes as measured on a Sun Ultra 2 Model 2170. Note that all classes are treated independently and can be learned in parallel. Thus the response time is realistic and corresponds to the real evaluation time on a network of machines. For the reported experiments we have used one process per class, except for the letter domain for which we used two processes per class. The results are summarized in Table 8.2. In all domains, except in the letter domain, the rule language of DOGMA was restricted to $\mathcal{L}_m^1$ since this is the default used by C4.5. For the letter domain we used the subset option of C4.5 and the full rule language $\mathcal{L}_m$ of DOGMA. Also, as the theory weight for the MDL encoding we used 0.5, which is the default used by C4.5.

Domain	Size	Attr	Classes	Method	Time
Breast cancer	286	9	2	10-CV	7.0
Chess	3196	36	2	train 2130, test 1066	76.7
Congress vote	435	16	2	10-CV	1.9
Letter	20000	16	26	train 13333, test 6667	317.7
Lymphography	148	18	4	10-CV	2.1
Splice60	3190	60	3	train 2190, test 1000	321.6
Splice12	3190	12	3	train 2190, test 1000	116.7
Tic-Tac-Toe	958	9	2	10-CV	10.3

Table 8.1: The datasets used in the evaluation. The Size column gives the total size of the dataset. The Attr and Classes columns show how many attributes and classes the domains have. The Method column indicates whether a single training and test set or 10-fold crossvalidation was used as the evaluation method. For the train and test method we give also the training/test set sizes. Finally, the Time column gives the average response time of DOGMA, measured in CPU-minutes on a Sun Ultra 2, Model 2170.

As can be seen from Table 8.2, the final rules produced by DOGMA have consistently shorter description lengths than the corresponding C4.5 rules. In this set of domains DOGMA shortens the original rules on average by over 15%[1]. Although the improvement in accuracy is not very significant in a single domain, the results are at least indicative since the accuracy is improved in all domains except in Breast Cancer. Also the number of rules tends to drop, especially when there are many rules, although there is more variation than in the description lengths.

A word about the response times of DOGMA. The evaluation time of

[1] The MDL for C4.5 in Congress vote and Lymphography are quite high, due to the fact the C4.5 does not always produce rules for all classes.

Domain	IA	FA	IR	FR	IL	FL
Breast cancer	70.3±7.8	68.2±13.0	10.7	10.4	434.6	432.6
Chess	98.9	99.4	23.0	21.0	452.9	378.0
Congress vote	95.8±2.0	96.3±1.1	6.0	7.9	431.7	220.5
Letter	76.2	77.0	248.0	214.0	36561.9	36051.6
Lymphography	77.7±7.4	80.4±6.5	9.7	12.2	226.1	187.3
Splice60	92.3	94.5	89.0	71.0	1847.7	1830.2
Splice12	93.1	94.6	90.0	70.0	1605.6	1592.3
Tic-tac-toe	98.5±1.9	99.5±0.7	22.0	20.0	288.9	175.5

Table 8.2: Results from eight UCI domains. The IA and FA columns stand for the test set accuracy percentages of the initial C4.5Rules theory and of the final theory. IR, FR, IL, and FL show the number of rules and description lengths of the initial vs. final theory. For the crossvalidation runs we show also the standard deviations of the accuracies.

DOGMA is affected by the size of a domain, both in terms of the number of examples and the number of attributes. Another central factor in evaluation time is the disjunctivity of the concept, i.e. how many rules are needed for the concept description. As the number of rules increases, also the number of species grows and, consequently, the GA populations get bigger. In the experiments reported in Table 8.2 we determined the population sizes by $15 \times (r + 3)$, where r is the number of background rules for the class. The evaluation times of C4.5Rules were very short (< 0.5 minute), except in the letter domain, which took 25.2 CPU-minutes.

A background rate parameter controls how much and how often the background rules are used in DOGMA. This parameter controls how many bits bits are in average copied from the background rules. When the background rate rises more bits are directly copied from the background rules, making them more dominant in the GA. In Figures 8.1 and 8.2 we show how different background rates affect the MDL of the best theory learned in each generation of the GA in the Tic-tac-toe and chess domains. In both domains DOGMA was run for 100 generations, with different background rates.

In the Tic-tac-toe domain, see Figure 8.1, the initial C4.5 rules are quite easily improved, even when no background seeding is used. Observe that the background rules are usually helpful, even when no background seeding is used, since they are used for setting suitable parameters for the MDL-based fitness function f_M. The best result for Tic-tac-toe is achieved with a 50

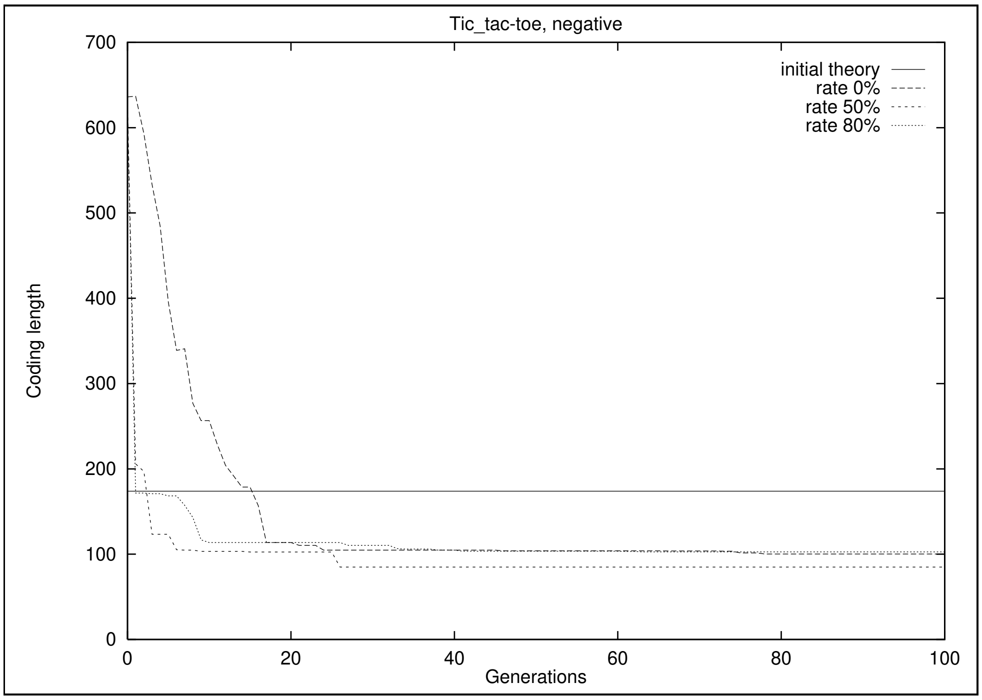

Figure 8.1: The MDL coding length of the best theory in each generation for different background rates in the Tic-tac-toe domain, class negative.

% background rate. The 80 % rate does slightly worse; this suggests that the initial rules become too dominant in this case and hinder the search of new, shorter theories. In the chess domain, see Figure 8.2, the initial rules are much better than in the Tic-tac-toe domain. Therefore, more work is needed in order to improve on them. In the Chess domain higher seeding rates are both faster and more efficient than the lower rates; this suggests that DOGMA can use large substructures from the initial rules.

8.3 Conclusions

In this chapter we have described how to use the GA-based theory revision system DOGMA to enhance rule-based classification theories. In our methodology we first use a fast concept learner to produce a first approximation of the data and then improve the result in DOGMA by building a second approximation of the data. The search in DOGMA is guided by an MDL-based fitness measure f_M. The results from 8 well-known machine learning domains indicate that DOGMA is able to improve on rules learned

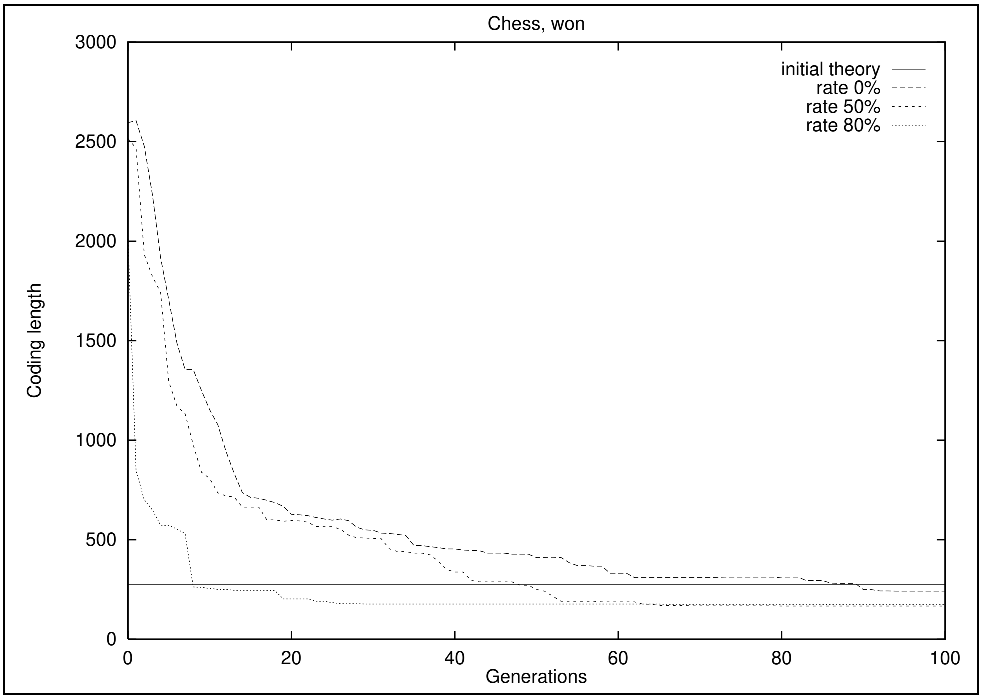

Figure 8.2: The MDL coding lengths of the best theory in each generation for different background rates in the Chess domain, class won.

by C4.5Rules, both with respect to the predictive accuracy and the description length. The methodology itself is of course not limited to C4.5 as a preprocessor or to f_M as a model selection criterion. We could in principle have used other rule learners and other fitness measures.

Chapter 9

Relational Learning

As we have emphasized throughout this thesis, DOGMA and JGA can be used both in propositional and relational domains. In this chapter we will concentrate on relational learning. For this aim we will give some general remarks regarding different learning settings and discuss the differences and similarities of propositional and relational learning. We will also discuss the relational setting used in DOGMA.

9.1 Relational Learning: General Remarks

9.1.1 Different Learning Settings

As we said in Chapter 2 there are several different settings for performing relational learning. Different formalizations include learning from interpretations [117, 24, 23] and learning from entailment [97, 86]. There are many open questions as to how exactly the different formalizations relate to each other. Therefore, it is still unclear how to compare relational systems which learn from entailment with those learning from interpretations. However, De Raedt [21] brings some light to the relationship between different logical settings for relational learning.

Following De Raedt we will define three different relational learning settings. We will also briefly discuss the differences between propositional and relational learning. More discussion of different relational learning settings can be found in [85, 37, 38]. The first relational learning setting we consider is learning from interpretations, where the examples are interpretations and an example is covered if it is a model of the hypothesis.

Definition 10 (Learning from interpretations) *Let T be a clausal* *theory and* $(e;\ class)$ *an example, where* e *is a Herbrand interpretation.* T *covers* e *under interpretation,* $e \in_i T$*, iff* e *is a model for* T*.*

In learning from interpretations the language of examples is the set of Herbrand interpretations of a given fixed alphabet. At first this might sound a bit confusing, but becomes more clear if we view each Herbrand interpretation as the set of true and false facts that it defines. This means that in the interpretational setting examples are required to be fully specified and there cannot be missing values in the examples. Propositional learning falls, in general, in the learning from interpretations setting. Learning from interpretations may therefore be regarded as the first order equivalence of traditional propositional learning. The second setting we consider is learning from entailment in which the examples are clauses and the covers relation is defined through semantic entailment.

Definition 11 (Learning from entailment) *Let* T *be a clausal theory and* $(e;\ class)$ *an example, where* e *is a clause.* T *covers* e *under entailment,* $e \in_e T$*, iff* $T \models e$*.*

Thus in the entailment setting the examples are more general than in the interpretational setting. It is also possible to have missing information in this setting since the antecedents of the clauses in examples may leave some predicates undefined. In the third setting the examples are made still more general by allowing them to be full clausal theories. The covers relation will then be defined through the joint satisfiability of the hypothesis and the example.

Definition 12 (Learning from satisfiability) *Let* T *be a clausal theory and* $(e;\ class)$ *an example, where* e *is a clausal theory.* T *covers* e *under satisfiability,* $e \in_s T$*, iff* $T \wedge e \not\models \Box$*.*

Thus learning from satisfiability seems to be the most general setting. Indeed, in [21] it is shown that learning from interpretations reduces to learning from entailment, which in turn reduces to learning from satisfiability. a reduces to b means that a learning problem in a can always be transformed into an equivalent learning problem in b. Unfortunately, the generality of a learning setting burdens it also computationally; in general it is more difficult to learn in a more general setting.

Many ILP systems take into account also background knowledge in the learning process. A special case of this is the extensional inductive logic programming setting used by many relational learners, e.g. FOIL [97] and

GOLEM [86], where the background knowledge is a set of true ground atoms and the examples are ground atoms. De Raedt shows that also extensional inductive logic programming reduces to learning from entailment. This setting is also known as the normal semantics for ILP or the explanatory setting. Also the interpretational setting may be augmented with a definite clausal theory B, serving as background domain knowledge. In this case it is often assumed that the example e is also a definite clause. Then B can be used to complete the example by turning it into a Herbrand interpretation. A few ILP systems, e.g. Claudien [23], ICL [25] and TILDE [14], implement this setting.

9.2 DOGMA as a Relational Learner

Propositional learners use, in general, learning from interpretations, whereas most relational learners perform learning from entailment. Knowing that our systems work in both propositional and relational domains, this raises the question of what kind of a learning setting do our systems apply. Both JGA and DOGMA learn according to the interpretational setting. To see this, consider a language template Λ and an example $e = o_1, \ldots, o_n$, composed of n objects as in Chapter 3. The language template defines literals that can be instantiated through the object definitions in e. Now, this can equivalently be viewed as the set of ground atoms that e defines in the language $\mathcal{L}$ defined by Λ. Such a set of ground atoms corresponds to a Herbrand interpretation and a formula $\phi \in \mathcal{L}$ matches e iff the Herbrand interpretation corresponding to e is a model of ϕ. Also some other GA-based relational learners like REGAL and G-NET use an interpretational setting. Learning from interpretations reduces to learning from entailment and most ILP systems use the latter setting. Our framework is therefore representationally weaker than that of most ILP systems. On the other hand, some recent ILP systems like Claudien and TILDE use learning from interpretations. Also, as is pointed out by De Raedt [21] some learning problems are more easily solved in this setting.

DOGMA's use of background knowledge differs substantially from other learning systems. Actually, in many domains DOGMA would simply encode ILP style background knowledge in the examples and in the language template. Extensional ILP systems use background knowledge to define the examples as implications. Interpretational ILP systems again use background knowledge to build the examples into interpretations. In both cases the validity of a formula is measured based on the examples expanded with the background knowledge. The background knowledge in ILP thus affects the validity of a formula. Consequently, background knowledge is assumed to be (approximately) correct. In DOGMA this is not the case; background knowledge is just a pool of knowledge that may be used in learning. There is no assumption of correctness associated with it. Also the background knowledge is not used totally as in ILP, but instead different fragments of it is tried out. Background seeding in DOGMA is stochastic, it considers different parts of background rules and tries to adapt them on different examples. If a part has a successful adaptation, it survives and contributes to the learning process, otherwise it is forgotten until it is seeded again. The task of DOGMA is to try out different fragments of background knowledge and to seek out which ones are good and how they can be used to classify examples.

Chapter 10

Evaluation of Relational Learning

To show how DOGMA works on relational domains we conduct three experiments on different relational problems. Each problem highlights different aspects of relational learning. As a gentle introduction, the first problem reviews a restricted relational application, where the description language has only one variable. The examples, on the other hand, consist of several objects. The second problem is a full-fledged relational problem, i.e. both the description language and the examples are relational. The third problem tests how DOGMA reacts to different kinds of noises in a relational setting.

10.1 Musk

A multiple instance problem is a restricted relational problem, where the examples consist of several unrelated objects and the language that describes the concept is only allowed to refer to a single variable, i.e. the concept description language is actually propositional. These kinds of problems appear, for example, in propositional learning when the examples are ambiguous, so that a single object is actually represented as a set of possible alternatives. Multiple instance problems occur frequently in biochemistry, e.g. in drug activity prediction, where one wants to determine if a specified property holds for some molecules. The problem, however, is that many molecules have several possible low-energy conformations or alternative shapes. Therefore, the prediction task is a multiple instance problem that infers whether one of the possible conformations has the specified property.

In [28] Dietterich, Lathrop, and Lozano-Pérez investigate how several learning algorithms perform in the musk-odor prediction task. The domain consists of 92 molecules, which are divided into 47 musk molecules and 45 non-musk molecules according to judgments made by human experts. The task is to learn to predict to which class a molecule belongs. However, since the molecules have several possible conformations, the problem involves inference over multiple instances. The dataset is built by generating the low-energy conformations of a molecule, after which the conformations are filtered so that some highly similar ones are removed. Together the 92 molecules have 476 conformations in the dataset. Each conformation is described by 166 features. The 162 first features measure the distances, in hundredths of Ångströms, from a molecule's origin to its surface along different rays. Therefore, these features roughly describe the shape of the conformation. The last four features represent the position of a designated oxygen atom on the surface of the molecule.

Since the features have continuous values, we have discretized the data with the entropy-based discretize utility in the $\mathcal{MLC}$++ package [66]. In the discretization we ignored the multiple instance problem and treated each conformation as a separate example. In general, as pointed out by Blockeel and De Raedt [13], there are problems in using a propositional discretizer in a relational domain. Therefore, the discretization method we have applied is not optimal. Nevertheless, the learning results obtained in this domain are good, so even a propositional discretizer seems to perform reasonably well. The learning setup that we have used in the musk domain is described in Table 10.1. The fitness function we use is f_{MG}. As a language bias we restrict the hypotheses to 1-CNF rules, i.e. we apply the language $\mathcal{L}_m^1$. Since there are only 92 examples in the domain, we have applied 10-fold crossvalidation to derive a more accurate approximation of the performance of DOGMA$_{MG}$. Notice that each training fold in the crossvalidation must be discretized separately.

The results are summarized in Table 10.2, where we confront our results with the ones obtained by Dietterich et al. in [28]. We have also included in Table 10.2 a result from [14], where the relational decision tree learner TILDE was applied to the musk prediction task. TILDE was applied to the problem with different discretizations and language biases, so the result we report here is the best result obtained in [14]. Dietterich et al. hypothesized that a suitable representation language in the musk domain consists of so-called axis-parallel hyper-rectangles (APR). A hypothesis in this language simply specifies bounds for the molecule's surface along the rays. In our parlance the representation language is similar to $\mathcal{L}_1^c$, i.e. the conjuncts in a rule specify features with bounds specified by contiguous intervals. Diet-

Predicates	Comment
$f1(X, [Val_{0_1}, \ldots, Val_{n_1}])$	The predicate for feature 1, i.e. the distance of the conformation from the origin to the molecules surface along ray 1. $Val_{0_1}, \ldots, val_{n_1}$ are the intervals produced by the discretizer.
$\vdots$	$\vdots$
$f166(X, [Val_{0_{166}}, \ldots, Val_{n_{166}}])$	The predicate for feature 166 relating to the position of a designated oxygen atom. $Val_{0_{166}}, \ldots, val_{n_{166}}$ are the intervals produced by the discretizer.

Language Template
$f1(X, [Val_{0_1}, \ldots, Val_{n_1}]) \wedge \ldots \wedge f166(X, [Val_{0_{166}}, \ldots, Val_{n_{166}}])$

Table 10.1: The language definition for learning musk-odor molecules.

terich et al. designed several special APR algorithms, some of which took the multiple instance problem into consideration and compared their performance to a backpropagation neural network and to C4.5. As the last two are propositional learners they could not utilize the multiple conformations and were therefore run treating each conformation as a separate example. The APR algorithms that can take the multiple instance problem into account are marked with an $*$ in Table 10.2.

As can be seen from the results in Table 10.2, the general propositional learners that cannot handle the many-to-one problem of conformations vs. molecules perform poorly. DOGMA makes totally 9 prediction mistakes in the crossvalidation and is thus among the top four algorithms. As can be seen from the confidence intervals, DOGMA and the best three APR algorithms have quite similar error probabilities. This shows that relational algorithms can be directly and efficiently used in solving multiple instance problems. The other relational algorithm in Table 10.2, TILDE, performs also quite well although slightly worse than DOGMA and the top three APR algorithms. The success of the APR algorithms can, at least partially, be explained by the fact that they are specialized, both through their language bias and their search strategy, to this problem.

Algorithm	Errors	Performance%
*Iterated-discrim APR	7	92.4[87.0 – 97.8]
*GFS elim-kde APR	8	91.3[85.5 – 97.1]
DOGMA$_{MG}$	9	90.2[84.2 – 96.3]
GFS elim-count APR	9	90.2[84.2 – 96.3]
TILDE	12	87.0[80.1 – 93.8]
GFS all-positive APR	15	83.7[76.2 – 91.2]
All-positive APR	18	80.4[72.3 – 88.5]
Backpropagation	23	75.0[66.2 – 83.8]
C4.5	29	68.5[40.9 – 61.3]

Table 10.2: Results from a 10-fold crossvalidation in the musk-odor domain. We report the number of errors of each algorithm on the 10 folds and the average percentages of correct predictions. For the predictions we show also the 95% confidence intervals.

10.2 Thousand Trains

In 1980 Michalski [76] proposed a relational learning problem involving different kinds of trains going east or west, where each train is made up of number of coaches. Inspired by this Giordana and Saitta composed a similar learning problem [44]. Unlike Michalski, who used a very small learning set of 10 trains, Giordana and Saitta composed a set of 1000 trains. The task is to discriminate between eastbound and westbound trains based on the description of the trains. The set of 1000 trains has been randomly generated so that there are 500 trains of both classes. A train is classified as an eastbound train if the following rules apply:

Rule1 In the second position there is an open-top small coach, carrying one load, followed by an open-top small coach.

Rule2 In the third position there is a closed-top small coach, carrying one load and in the fifth position there is an open-top small coach.

Rule3 In position two, three or four there is a small coach, with two wheels and carrying one load, immediately followed by a long white coach carrying one load.

The challenge of the learning system is to learn these rules, or equally effective ones, from the training data. This problem has also been attacked by

Predicates	Comment
$position(X, [0, 1, 2, 3, 4])$	The position of the coach, counting from the engine.
$length(X, [1, 2])$	Length of the coach.
$wheels(X, [2, 3])$	The number of wheels of the coach.
$nloads(X, [0, 1, 2, *])$	Number of loads.
$color(X, [yl, wh, rd, gr, gy, bk])$	Color of the coach.
$shape(X, [en, ot, or, us, dr, he, el, cr, jt])$	Shape of the coach, en = engine, ot = open-trap, etc.
$distance(X, Y[1, 2, *])$	The distance between coaches X and Y.

Constraint Predicates	Comment
$coach(X)$	X is a coach, i.e. X is not an engine.
$follows(X, Y)$	Y comes after X.

Language Template
$coach(X) \wedge position(X, [0, 1, 2, 3, 4]) \wedge length(X, [1, 2]) \wedge$ $wheels(X, [2, 3]) \wedge nloads(X, [0, 1, 2, *]) \wedge color(X, [yl, wh, rd, gr, gy, bk])$ $\wedge shape(X, [en, ot, or, us, dr, he, el, cr, jt]) \wedge$ $coach(Y) \wedge position(Y, [0, 1, 2, 3, 4]) \wedge length(Y, [1, 2]) \wedge$ $wheels(Y, [2, 3]) \wedge nloads(Y, [0, 1, 2, *]) \wedge color(Y, [yl, wh, rd, gr, gy, bk])$ $\wedge shape(Y, [en, ot, or, us, dr, he, el, cr, jt]) \wedge$ $position(Z, [0, 1, 2, 3, 4]) \wedge length(Z, [1, 2]) \wedge wheels(Z, [2, 3]) \wedge$ $nloads(Z, [0, 1, 2, *]) \wedge color(Z, [yl, wh, rd, gr, gy, bk]) \wedge$ $shape(Z, [en, ot, or, us, dr, he, el, cr, jt]) \wedge$ $position(Z, [0, 1, 2, 3, 4]) \wedge length(Z, [1, 2]) \wedge wheels(Z, [2, 3]) \wedge$ $nloads(Z, [0, 1, 2, *]) \wedge color(Z, [yl, wh, rd, gr, gy, bk]) \wedge$ $shape(Z, [en, ot, or, us, dr, he, el, cr, jt]) \wedge$ $follows(X, Y) \wedge distance(X, Y, [1, 2, *]) \wedge distance(Y, Z, [1, 2, *]) \wedge$ $distance(X, Z, [1, 2, *])$

Table 10.3: The language definition in DOGMA for learning eastbound trains.

the GA-based learning system REGAL [42, 88]. We use the same knowledge representation as REGAL, the setup in Table 10.3 is from [88].

The language template in Table 10.3 defines a learning problem involving three variables, i.e. the task is to learn $eastbound \leftarrow \Phi(X, Y, Z)$ given the available predicates. The conditions open-top and closed-top in the rules correspond actually to the following sets of values $\{ot, or, us, dr\}$ and $\{en, he, el, cr, jt\}$, respectively.

Each example train is represented as a set of coaches, where each coach is described by its position, shape, color, length, and by the number of wheels and loads. An example train is shown in Table 10.4.

class	position	shape	color	length	# wheels	# loads
eastbound	0	en	wh	2	2	0
	1	el	wh	1	2	1
	2	or	wh	2	2	1
	3	cr	gr	1	2	0
	4	yt	yl	2	2	1

Table 10.4: An example train with an engine and four coaches

We learned the class of eastbound trains in DOGMA both with the set covering fitness f_{SC} and with the combined MDL and gain fitness f_{MG}. In both cases DOGMA learned complete and consistent descriptions of the eastbound train concept. When f_{SC} was used the resulting description consisted of three rules, whereas DOGMA with f_{MG} learned a complete and consistent description using only two rules. As an example of the rules learned by DOGMA for the class of eastbound trains, we show in Table 10.5 the result of DOGMA$_{MG}$. REGAL has also been able to learn complete and consistent rules in this domain. Depending on the version of the system [46, 42, 88], REGAL has learned description with 5 or 8 rules. Also G-NET [41] has been applied to this task and has learned a classification theory composed of two rules which are very similar to the ones described in Table 10.5.

10.3 Relational Chess

As a further demonstration of how DOGMA is applied to relational learning we use a relational domain taken from the game of chess. The task is to learn the concept of an illegal white-to-move situation in a chess end-game with a white king and rook versus a black king. Unlike the KRK chess problem

Rule	Cover
$coach(X) \wedge length(X,[1]) \wedge load(X,[1]) \wedge$ $coach(Y) \wedge length(Y,[2]) \wedge load(Y,[1,3]) \wedge color(Y,[wh])$	(216 0)
$coach(X) \wedge length(X,[1]) \wedge load(X,[1]) \wedge$ $shape(X,[us,or,cr,el,yt,he,sl]) \wedge$ $coach(Y) \wedge length(Y,[1]) \wedge shape(Y,[ot,us,or,yt]) \wedge$ $position(Z,[0,2,3]) \wedge load(Z,[1]) \wedge distance(X,Y,[2])$	(309 0)

Table 10.5: Rules for eastbound trains found by DOGMA$_{MG}$.

introduced in Chapter 6 this problem is formulated in relational terms. This has been a popular example domain in the ILP literature [84, 97, 69, 32], mainly because the data has also versions which are corrupted by different types of noise. Therefore, this domain may be used to test the noise-handling capabilities of relational learning systems.

10.3.1 ILP Representation

In the original ILP formulation [84, 32] the problem is presented with a six-place predicate $illegal(WKf, WKr, WRf, WRr, BKf, BKr)$. The target predicate states that the positions of the white king and rook at (WKf, WKr) and (WRf, WRr) and the position of the black king at (BKf, BKr) do not define a legal white-to-move configuration on the chessboard. For example, consider the literal

$$illegal(8,5,5,4,1,4), \tag{10.1}$$

it represents the configuration on the chessboard, where the white king is in position $(8,5)$, the white rook in position $(5,4)$, and the black king in position $(1,4)$. This is an illegal white-to-move position since the white rook threatens the black king.

The background knowledge is represented either by two or four relations. In FOIL [97] the relations $adjacent(X,Y)$ and $less\_than(X,Y)$ are used. These indicate that the file and/or rank of X is adjacent to or less than the file and/or rank of Y. In many other ILP systems, e.g. in the mFOIL system [32], the corresponding relations are typed and the background knowledge consists of the relations $adjacent\_file(X,Y)$, $adjacent\_rank(X,Y)$, $less\_than\_file(X,Y)$, and $less\_than\_rank(X,Y)$. Typing means, for example, that the arguments of $less\_than\_rank(X,Y)$ can be only of type rank. In addition many systems use a built-in equality relation.

Thus the ILP representation of the illegal white-to-move problem consists of examples of the form

$$\langle illegal(x1, x2, y1, y2, z1, z3), class\rangle,$$

where the class is either $\oplus$ or $\ominus$, indicating a positive versus a negative example. The background knowledge consists of definitions of the above mentioned relations.

10.3.2 DOGMA representation

In DOGMA the problem is posed as $illegal \leftarrow \Phi(WK, WR, BK)$, where WK, WR, and BK are variables that stand for the white king, the white rook, and the black king. Each example is made of three objects that describe the positions of the three chess pieces. For example, the chessboard configuration of (10.1) would be represented by the example in Table 10.6.

class	position	file	rank
$\oplus$	0	8	5
	1	5	4
	2	1	4

Table 10.6: An example configuration of the chessboard represented in DOGMA

Just as in the ILP case a part of the task definition in DOGMA is the definition of the relevant learning predicates. We define the same four predicates as used by mFOIL: $adjacent\_file, adjacent\_rank, less\_than\_file$, and $less\_than\_rank$. We also define two equality predicates eq_file and eq_rank. The definition is completed by the language template in Table 10.7. The language template simply allows all suitable combinations of the defined predicates. Once again, we chose f_{MG} as the fitness function.

10.3.3 Experiments

We performed the same experiments as in [32], where FOIL and mFOIL were tested on uncorrupted chess examples as well as with data that was corrupted by different levels of noise.

In the uncorrupted case the systems were trained on five small training sets, each containing 100 examples. The classification accuracy was then tested on 5000 randomly generated examples. The distribution of positive

Predicate	Comment
$less\_file(X, Y, [y, n])$	The file of X is less than the file of Y.
$less\_rank(X, Y, [y, n])$	The rank of X is less than the rank of Y.
$adj\_file(X, Y, [y, n])$	The file of X is adjacent to the file of Y.
$adj\_rank(X, Y, [y, n])$	The rank of X is adjacent to the rank of Y.
$eq\_file(X, Y, [y, n])$	The file of X is equal to the file of Y.
$eq\_rank(X, Y, [y, n])$	The rank of X is equal to the rank of Y.
Constraints	**Comment**
$white\_king(X)$	X is a white king.
$white\_rook(X)$	X is a white rook.
$black\_king(X)$	X is a black king.

Language Template
$white\_king(X) \wedge white\_rook(Y) \wedge black\_king(Z) \wedge$
$less\_file(X, Y, [y, n]) \wedge less\_rank(X, Y, [y, n]) \wedge adj\_file(X, Y, [y, n]) \wedge$
$adj\_rank(X, Y, [y, n]) \wedge eq\_file(X, Y, [y, n]) \wedge eq\_rank(X, Y, [y, n]) \wedge$
$less\_file(X, Z, [y, n]) \wedge less\_rank(X, Z, [y, n]) \wedge adj\_file(X, Z, [y, n]) \wedge$
$adj\_rank(X, Z, [y, n]) \wedge eq\_file(X, Z, [y, n]) \wedge eq\_rank(X, Z, [y, n]) \wedge$
$less\_file(Y, Z, [y, n]) \wedge less\_rank(Y, Z, [y, n]) \wedge adj\_file(Y, Z, [y, n]) \wedge$
$adj\_rank(Y, Z, [y, n]) \wedge eq\_file(Y, Z, [y, n]) \wedge eq\_rank(Y, Z, [y, n])$

Table 10.7: The learning setup for the illegal white-to-move chess end-game problem in DOGMA.

and negative examples is, respectively, 1/3 and 2/3. The results shown in the first row of Table 10.8 are averages over the five sets.

Also in the corrupted tests we used the datasets provided by Džeroski. They introduced noise at different levels to the five small training sets. The large test set of 5000 examples is not affected by noise. Three kinds of noise were considered: noise only in the arguments, noise only in the class variable, and noise in both the arguments and the class variable. The noise levels were $5\%, 10\%, 15\%, 20\%, 30\%, 50\%$, and 80%. Here, a noise level of $p\%$ for argument A means that in $p\%$ of the examples the value of A was corrupted by replacing it by a random value from the uniform distribution of possible values. Background knowledge, i.e. the defined learning predicates, were not corrupted by noise.

We summarize the results in Table 10.8 and in Figures 10.1, 10.2, and 10.3. When only noise in arguments is considered, see Figure 10.1, we notice that DOGMA exhibits the best performance while the noise level is under 20%. When the noise level rises to 50% or more mFOIL outperforms both

NT	Arguments			Class			Arguments and class		
NL	D	F	m	D	F	m	D	F	m
0%	97.31	94.82	95.57	–	–	–	–	–	–
5%	94.36	91.64	91.87	92.47	94.23	94.26	92.75	89.28	89.64
10%	88.70	80.45	84.59	90.64	90.07	92.02	80.23	76.94	80.79
15%	83.37	76.81	80.06	88.86	88.31	89.96	78.07	72.88	76.58
20%	80.86	73.79	76.49	88.77	87.53	88.37	74.16	71.02	74.74
30%	72.72	71.19	74.04	87.14	84.54	86.01	67.08	66.14	69.65
50%	61.84	63.25	68.64	77.13	71.86	79.20	56.20	61.33	66.27
80%	56.42	57.58	66.30	53.72	60.65	65.04	52.85	57.88	61.73

Table 10.8: Results in the domain of relational chess. Here NT refers to the type of noise introduced, whereas NL stands for the level of the noise. D, F, and m stand, respectively, for DOGMA, FOIL, and mFOIL. The results show the predictive accuracy percentages of each system under different kinds of noise.

DOGMA and FOIL. Figure 10.2 shows the accuracies of the systems when noise is introduced in the class variable. In this case all the systems perform equally well up to and including the noise level of 30%. For the highest noise level mFOIL performs best. Similarly, when both the arguments and the class variable are noisy, see Figure 10.3, mFOIL learns the best rules for the very noisy sets (at least 50% noise). For noise levels of 20% or less DOGMA is slightly better than mFOIL. Overall, we notice that noise in the class variable alone has the smallest effect on the learning performances of the systems. Learning is most difficult when both the arguments and the class variable are noisy. Another fact that should be brought forward is that for the higher noise levels the accuracies of all systems degrade close to or even below the accuracy of the majority learner (recall that 66.7% of the examples are negative, provided that the class variable is uncorrupted). In other words, in the very noisy cases all the systems are really unable to learn a meaningful description of the concept.

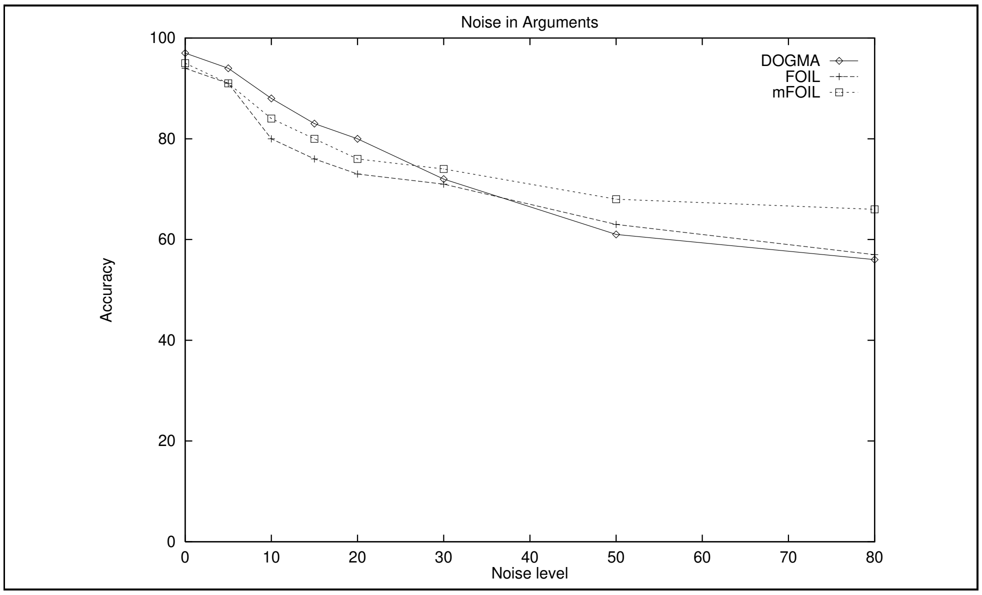

Figure 10.1: Results in the relational chess domain with various levels of noise in arguments.

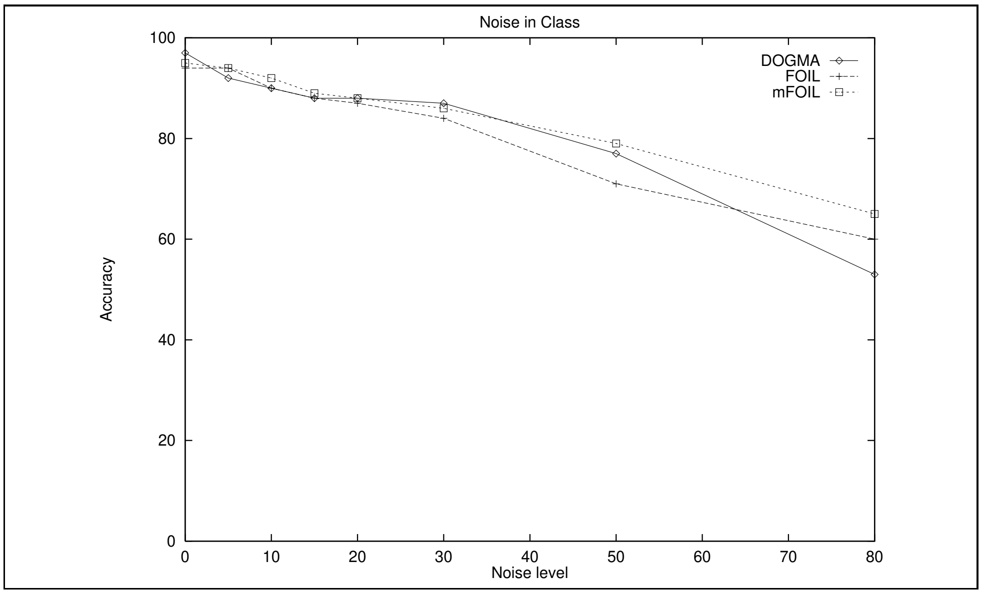

Figure 10.2: Results in the relational chess domain with various levels of noise in the class variable.

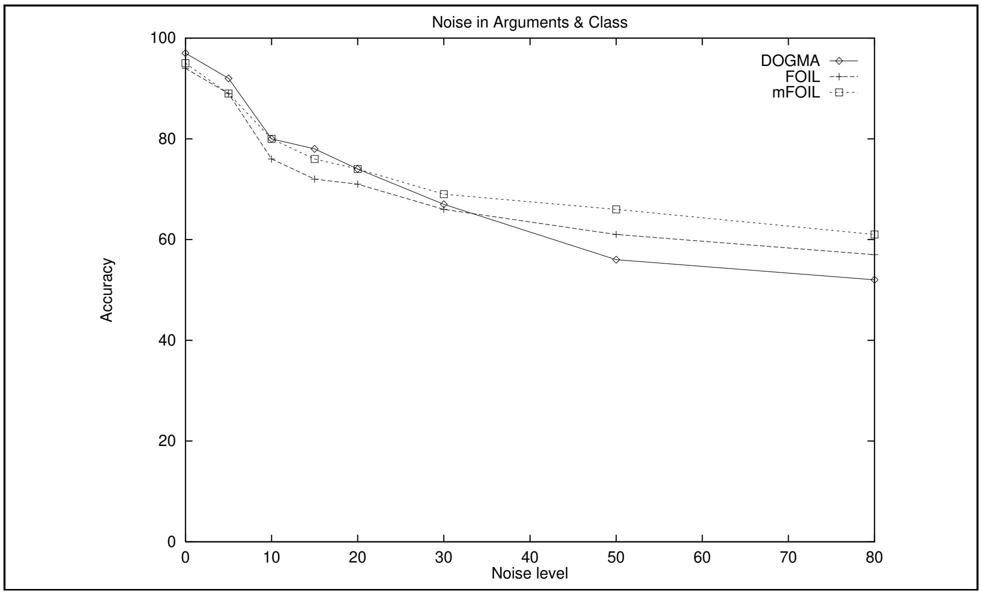

Figure 10.3: Results in the relational chess domain with various levels of noise in both arguments and class variable.

Chapter 11

Conclusions

11.1 Summary

In this thesis we have shown how GAs can be applied to rule-based concept learning and theory revision. We have discussed the underlying design philosophy of JGA, a GA-based system for concept learning. The system can be applied both in propositional and relational domains. We use a similar representation language as Michalski in his INDUCE system [77] and our learning setting corresponds to the one used in the REGAL system [44, 42]. However, our approach deviates from the one pursued in REGAL with respect to the methods used for learning disjunctive concepts. We deal with disjunctive concepts by using a novel cooperative scheme that combines features from both the Pittsburgh and Michigan approaches. We also include several different fitness functions in JGA, including two functions utilizing the MDL principle.

In DOGMA we have extended the symbiotic methodology of JGA to new areas of theory revision and learning using background knowledge. Unlike most other theory revision systems, DOGMA has no distance function that would compare the old theory and the new revised theory. Neither does DOGMA revise the old theory through a chain of small local changes. Instead, the system builds a completely new theory by stochastically sampling parts from the old theory and by combining them in a GA-based cooperative scheme. Our approach is computationally inspired. Using less computation and more of the initial theory leads to a theory that is similar to the original one and is only moderately adapted to the examples. Increasing the amount of computation allows for a more accurate adaptation to the examples and leads to a theory that is less dependent of the original theory.

We have evaluated our systems empirically and shown that they can be applied to both propositional and relational problems. The results in Chapters 6, 7, 8, and 10 reveal that JGA and DOGMA do find rules that are in most cases at least as accurate as the classifiers produced by other well-known systems. In Chapter 6 we established that JGA_{MG} does, in the tested cases, learn more accurate descriptions than three well-known learners: CN2, AQ15 and both rule and tree versions of C4.5. One should, however, be cautious when interpreting these results since JGA_{MG} was given a distinct advantage in the form of parameter tuning. Nevertheless, the results reveal that JGA_{MG} with parameter tuning is a robust concept learner that learns comparatively accurate descriptions. In Chapter 8 we evaluated the rule enhancement capabilities of DOGMA by improving rules produced by C4.5Rules. Again, the results show that DOGMA is able to enhance the initial C4.5 rules, both with respect to the MDL measure and the predictive accuracy. Unlike in Chapter 6, where due parameter tuning JGA might have been optimizing more suitable model selection criteria than C4.5Rules, in this case DOGMA and C4.5Rules are optimizing the same MDL-based model selection criterion. Overall, JGA and DOGMA are competitive compared to other greedier concept learners with respect to the predicative accuracy of the learned classifiers.

11.2 Future Work

A run with a GA usually involves user input in the form of parameter settings. Our systems are not exceptions in this respect. To apply the systems to a particular learning problem one has to specify numerous parameters for the GA-operators and the fitness functions. However, most of the parameters have remained unchanged throughout the experiments performed in this thesis, so they can be considered constants. The parameters that have undergone changes include important characteristics like the population size, the number of species, the seed generality, and various fitness parameters. However, when we use DOGMA for rule enhancement with the MDL-based fitness function f_M, most of these parameters are set automatically. We are currently developing methods that allow the same to be done also in other cases, when there is no background theory available. This can be done by adapting the parameter values to the examples during the initial phase of a run. Self-adaptation in GAs has been studied, for example, by Bäck in [7]

We have applied our systems to domains having several hundreds of features, resulting in bitstrings with several thousands of bits. We have also learned descriptions from large example sets including tens of thousands of examples. This shows that the systems can be applied to practical problems of considerable size. However, to attack huge problems, common in the field of data mining, consisting of hundreds or thousands of features and hundred thousands or millions of examples, we have to extend the systems with new methods. Future work in this area includes sampling, to cope with very large datasets, and automatic feature selection that can be applied to reduce the dimensionality of a learning domain.

JGA and DOGMA learn general rules that (approximately) discriminate a concept class from the other classes. In the field of knowledge discovery from databases and data mining there is considerable interest to learn rules that have a more characteristic nature. Such rules are more special than the discriminant ones and aim at characterizing certain properties of the concept. As pointed out in [23, 43], the logical setting used by us can also be extended to characteristic learning. Our systems could also be applied to this task, given a fitness function that promotes the search of characteristic rules.

Appendix A

The parameters and evaluation times of JGA in Chapter 6

The parameters of JGA in the runs reported in Chapter 6 were set by dividing the training set into two parts and then using one part for learning and the other part for testing the suitability of the parameters. To avoid overfitting of the parameters we performed only 2-3 such initial parameter tuning runs. In the case of crossvalidation the parameters were tuned in a similar fashion only for one of the training folds. Although JGA has many parameters, for most of them we always use the default values, and they act therefore as constants.

In Table 1 we show the main parameters that affect the time complexity of JGA. In some domains different classes had different parameters. In such cases we show the minimum and maximum values of the parameters. The number of species gives an upper bound on the number of rules in the families. However, normally JGA learns a more compact classification theory. For example, in the Mushrooms domain, where the number of species is set to 8, JGA learns three rules for poisonous mushrooms.

In addition to the parameters, we also show the response time of JGA in each domain. The response time is defined as $T = max(T_{ij})$, where T_{ij} is the evaluation time of process j for concept class i. In other words the response time is the maximum time of all the processes running a GA-population for a class. For the crossvalidation runs we report the maximum response time of the folds. The time is expressed in CPU-minutes as measured on a Sun Ultra 2 Model 2170. Notice that all classes are treated independently and can be learned in parallel. Therefore, the response time is realistic and corresponds to a real evaluation time on a network of machines.

Domain	Version	Populations	Gens	Species	Time
Mushrooms	SC	2x100	100	8	69.6
	MG	2x100	100	8	50.3
Chess	SC	2x720	100	30	120.6
	MG	2x720	100	12	125.0
Splice60	SC	4x500	300	16	632.4
	MG	4x240	120	10–12	191.6
Splice30	SC	4x600	150	16	295.4
	MG	4x600	150	12	264.8
Splice12	SC	4x500	100	16	50.0
	MG	4x500	100	10	66.6
Receptor170	SC	2x100–2x240	20–100	4–12	167.2
	MG	100–360	20–60	4–12	175.7
Receptor70	SC	2x50–2x180	20–100	4–12	53.1
	MG	100–360	20–100	4–12	120.0
Receptor28	SC	2x100–2x240	20–100	4–12	26.6
	MG	100–240	20–100	4–12	24.3
Australian	SC	700	150	35	43.5
	MG	210	80	7	7.5
Soybean	SC	150–300	60–80	5	2.4
	MG	150–300	60–80	5	6.5
Lymphography	SC	60–120	40–80	3–6	1.7
	MG	60–120	40–80	3–6	1.1

Table 1: The JGA parameters affecting time complexity. The parameter shown are: the number of populations, the size of the populations, the number of generations, and the number of species. In addition we show the response time of JGA in each domain.

Bibliography

[1] A. Agresti. *Analysis of Ordinal Categorical Data.* John Wiley & Sons, New York, 1984.

[2] L. Altenberg. The schema theorem and Price's theorem. In L. D. Whitley and M. Vose, editors, *Proceedings of Foundations of Genetic Algorithms, volume 3*, pages 23–49. Morgan Kaufmann, 1994.

[3] C. Angalano, A. Giordana, G. Lo Bello, and L. Saitta. An experimental evaluation of coevolutive concept learning. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 19–27, Madison, Wisconsin, 1998. Morgan Kaufmann.

[4] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, Cambridge, 1992.

[5] A. Aspnäs, V. Cockcroft, and J. Hekanaho. Use of genetic algorithms to learn ligand recognition concepts: Application to the GPCR superfamily. In *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*, pages 131–149, 1996.

[6] S. Augier, G. Venturini, and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pages 21–26, Montreal, Canada, 1995. AAAI Press.

[7] T. Bäck. Self-adaptation in genetic algorithms. In *Proceedings of the 1st European Conference on Artificial Life*, pages 263–271, Paris, France, 1992. MIT Press.

[8] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 57–62, Piscataway, NJ, June 1994. IEEE Press.

[9] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York, 1997.

[10] T. Bäck, U. Hammel, and H. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, April 1997.

[11] A. Barr and E. A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 1. William Kaufmann, Los Altos, CA, 1981.

[12] F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 305–317, Ann Arbor, 1988. Morgan Kaufmann.

[13] H. Blockeel and L. De Raedt. Lookahead and discretization in ILP. In *Proceedings of Inductive Logic Programming, 7th International Workshop*, pages 77–84, Prague, Czech Republic, September 1997. Springer.

[14] H. Blockeel and L. De Raedt. Top-down induction of logical decision trees. Technical Report CW 247, Dept. of Computer Science, K.U.Leuven, January 1997.

[15] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[16] G. Cestnik and I. Kononenko. Donors of the original lymphography dataset. 1988. Obtained from the University Medical Center, Institute of Oncology, Ljubljana, Yugoslavia.

[17] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[18] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 1995. Morgan Kaufmann.

[19] K. A. De Jong. *Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1975.

[20] K. A. De Jong, W. M. Spears, and F. D. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

[21] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.

[22] L. De Raedt and M. Bruynooghe. Interactive concept learning and constructive induction by analogy. *Machine Learning*, 8:107–150, 1992.

[23] L. De Raedt and L. Dehaspe. $\mathcal{C}$lausal $\mathcal{D}$iscovery. *Machine Learning*, 26:99–146, 1997.

[24] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[25] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Conference on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence, Vol 997*. Springer-Verlag, 1995.

[26] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, Fairfax, VA, 1989. Morgan Kaufmann.

[27] G. DeJong and R. Mooney. Explanation–based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[28] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pèrez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[29] S. K. Donoho and L. A. Rendell. Representing and restructuring domain theories: a constructive induction approach. *Journal of Artificial Intelligence Research*, 2:411–446, 1995.

[30] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202, Tahoe City, CA, July 1995. Morgan Kaufmann.

[31] G. Drastal, G. Czako, and S. Raatz. Induction in an abstraction space. In *Proceeding of the Eleventh International Joint Conference on Artificial Intelligence*, pages 708–712, Detroit, MI, 1989.

[32] S. Džeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the 4th Scandinavian Conference on Artificial Intelligence*, pages 111–125. IOS Press, 1993.

[33] W. Emde and D. Wettschereck. Relational instance-based learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 122–130, Bari, Italy, July 1996. Morgan Kaufmann.

[34] B. Everitt. *The Analysis of Contingency Tables*. Chapman and Hall, London, 1977.

[35] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence*, pages 1022–1027, San Mateo, CA, 1993. Morgan Kaufmann.

[36] S. E. Fienberg. *The Analysis of Cross-Classified Categorical Data*. MIT Press, Cambridge, Massachusetts, 1978.

[37] P. A. Flach. *Conjectures: an inquiry concerning the logic of induction*. PhD thesis, Institute for Language Technology and Artificial Intelligence, Tilburg University, Tilburg, Netherlands, 1995.

[38] M. Frazier and L. Pitt. Learning from entailment. In *Proceedings of the 10th International Workshop on Machine Learning*, Amherst, MA, 1993. Morgan Kaufmann.

[39] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. URL:http://www.netlib.org/pvm3/book/pvm-book.html.

[40] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1987.

[41] A. Giordana. Personal communication. 1998.

[42] A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416, 1995.

[43] A. Giordana, F. Neri, L. Saitta, and M. Botta. Integrating multiple learning startegies in first order logics. *Machine Learning*, 27:209–240, 1997.

[44] A. Giordana and L. Saitta. REGAL: An integrated system for learning relations using genetic algorithms. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 234–249, Harpers Ferry, VA, 1993.

[45] A. Giordana and L. Saitta. Stochastic search in concept learning. Invited talk at the Fourth International Workshop on Multistrategy Learning, 1998.

[46] A. Giordana, L. Saitta, and F. Zini. Learning disjunctive concepts with distributed genetic algorithms. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 115–119, Orlando, FL, 1994.

[47] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, San Mateo, CA, 1989.

[48] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93, San Mateo, CA, 1991. Morgan Kaufmann.

[49] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 41–49, Cambridge, MA, 1987. Lawrence Erlbaum Associates.

[50] D. F. Gordon and M. desJardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20:5–22, 1995.

[51] D. P. Greene and S. F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993.

[52] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60, Los Altos, CA, 1987. Morgan Kaufmann.

[53] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381, 1990.

[54] C. Harley, R. Reynolds, and M. Noordewier. Donors of the original promoters dataset. 1990.

[55] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40, 1989.

[56] J. Hekanaho. Symbiosis in multimodal concept learning. In *Proceedings of the 12th International Conference on Machine Learning*, pages 278–285, Tahoe City, CA, 1995. Morgan Kaufmann.

[57] J. Hekanaho. Background knowledge in GA-based concept learning. In *Proceedings of the 13th International Conference on Machine Learning*, pages 234–242, Bari, Italy, 1996. Morgan Kaufmann.

[58] J. Hekanaho. Testing different sharing methods in concept learning. Technical Report 71, TUCS Technical Report, December 1996. URL:http://www.tucs.fi/publications/techreports/.

[59] J. Hekanaho. GA-based rule enhancement concept learning. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 183–186, Newport Beach, CA, 1997. AAAI Press.

[60] W. D. Hillis. Co-evolving parasites improves simulated evolution as an optimization procedure. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Proceedings of Artificial Life II*, pages 313–324. Addison Wesley, 1992.

[61] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[62] J. H. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An AI Approach*, volume II, pages 593–623, San MAteo, CA, 1986. Morgan Kaufmann.

[63] R. Holte. Donor of the original chess dataset. 1989.

[64] R. C. Holte, L. E. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artifical Intelligence*, pages 813–818, Detroit, MI, 1989. Morgan Kaufmann.

[65] C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.

[66] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 740–743. IEEE Computer Society Press, 1994. URL: http://www.sgi.com/Technology/mlc/.

[67] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 114–119, Portland, Oregon, August 1996. AAAI Press.

[68] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[69] N. Lavrač and S. Džeroski. Inductive learning of relations from noisy examples. In S. Muggleton, editor, *Inductive Logic Programming*, pages 495–516. Academic Press, 1992.

[70] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

[71] J. W. Lloyd. *Foundations of Logic Programming.* Spriger-Verlag, 1984.

[72] S. W. Mahfoud. *Niching Methods for Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[73] R. A. McCallum and K. A. Spackman. Using genetic algorithms to learn disjunctive rules from examples. In *Proceedings of the 7th International Workshop on Machine Learning*, pages 149–152. Morgan Kaufmann, 1990.

[74] C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1996. URL: http://www.ics.uci.edu/~mlearn/-MLRepository.html.

[75] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 3rd edition, 1996.

[76] R. S. Michalski. Pattern recognition as a rule–guided inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:349–361, 1980.

[77] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 83–134, Palo Alto, CA, 1983. Tioga Press.

[78] R. S. Michalski and R. L. Chilausky. Knowledge acquisition by encoding expert rules versus computer induction from examples: a case

study involving soybean pathology. *International Journal of Man-Machine Studies*, 12:63–87, 1980.

[79] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application in three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, Philadelphia, PA, 1986.

[80] D. Michie, D. Spiegelhalter, and C. T. (Eds.). *Machine Learning, Neural Networks and Statistical Classification.* Ellis Horwood, London, 1994.

[81] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts, 1996.

[82] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[83] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation–based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[84] S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. Experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113–118. Morgan Kaufmann, 1989.

[85] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[86] S. Muggleton and C. Feng. Efficient induction in logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381, Ohmsha, Tokyo, Japan, 1990.

[87] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, San Mateo, CA, 1991.

[88] F. Neri. *First Order Logic Concept Learning by means of a Distributed Genetic Algorithm.* PhD thesis, University of Torino, Turin, Italy, 1997.

[89] F. Neri and L. Saitta. Exploring the power of genetic search in learning symbolic clasifiers. *IEEE Tranactions on Pattern Analysis and Machine Intelligence*, 18(11):1135–1141, 1996.

[90] D. Ourston and R. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *1990 National Conference on Artificial Intelligence*, 1990.

[91] J. Paredis. Co-evolutionary constraint satisfaction. In Y. Davidor, H. P. Schefek, and R. Manner, editors, *Proceedings of the* 3^{rd} *Conference on Parallel Problem Solving from Nature*, pages 46–55, Jerusalem, Israel, 1994. Springer.

[92] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–91, 1992.

[93] Z. Perry. *Experimental Study of Speciation in Ecological Niche Theory Using Genetic Algorithms.* PhD thesis, University of Michigan, Ann Arbor, MI, 1984.

[94] M. A. Potter, K. A. De Jong, and J. J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the 7th International Conference on Genetic Algorithms*, Pittsburgh, Pensylvania, USA, 1995. Morgan Kaufmann.

[95] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[96] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[97] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[98] J. R. Quinlan. Technical note: Improved estimates for the accuracy of small disjuncts. *Machine Learning*, 6:93–98, 1991.

[99] J. R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[100] J. R. Quinlan. The minimum description length principle and categorical theories. In *Proceedings of the 11th International Workshop on Machine Learning*, pages 233–241, New Brunswick, 1994. Morgan Kaufmann.

[101] J. R. Quinlan. MDL and categorical theories (continued). In *Proceedings of the 12th International Conference on Machine Learning*, pages 464–470, Tahoe City, CA, 1995. Morgan Kaufmann.

[102] D. Reynolds and J. Gomatam. Stochastic modelling of genetic algorithms. *Artificial Intelligence*, 82:303–330, 1996.

[103] B. L. Richards and R. J. Mooney. Refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[104] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[105] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, River Edge, NJ, 1989.

[106] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[107] J. Schlimmer. Donor of the original mushrooms dataset. 1987.

[108] M. Sebag. Distance induction in first order logic. In N. Lavrač and S. Džeroski, editors, *Proceedings of Inductive Logic Programming, 7th International Workshop*, pages 264–272, Prague, Czech Republic, September 1997. Springer.

[109] M. Sebag and C. Rouveirol. Tractable induction and classification in first order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence*, pages 888–892. Morgan Kaufmann, 1997.

[110] C. E. Shannon and W. Weawer. *The Mathematical Theory of Communication.* The Univeristy of Illinois Press, 1949.

[111] A. Shapiro. *The Role of Structured Induction in Expert Systems.* PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1983.

[112] S. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the 8th International Joint Conference on Artifical Intelligence*, pages 422–425, Karlsruhe, Germany, 1983. William Kaufmann.

[113] W. M. Spears. Simple subpopulation schemes. In A. V. Sebald and D. B. Fogel, editors, *Proceedings of the Third Conference on Evolutionary Programming*, pages 297–307. World Scientific, 1994.

[114] G. Towell, M. Noordewier, and J. Shavlik. Donors of the original splice-junction dataset. 1992.

[115] G. Towell and J. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[116] G. Towell and J. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165, 1994.

[117] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[118] G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attribute based concepts. In *Proceedings of the 7th European Conference on Machine Learning*, pages 280–296, Vienna, Austria, 1993.

[119] C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society (Series B)*, 49:240–252, 1987.

[120] C. S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.

[121] G. M. Weiss. Learning with rare cases and small disjuncts. In *Proceedings of the 12th International Conference on Machine Learning*, pages 558–565, Tahoe City, CA, 1995. Morgan Kaufmann.

[122] J. Wnek, K. Kaufman, E. Bloedorn, and R. S. Michalski. Selective induction learning system AQ15c: The method and user's guide. Technical report, Center for Machine Learning and Inference, George Mason University, March 1995.