

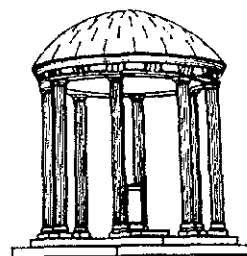
Inductive Learning of Decision Rules
from Attribute-Based Examples:
A Knowledge-Intensive
Genetic Algorithm Approach

TR91-030

July, 1991

Cezary Z. Janikow

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175



UNC is an Equal Opportunity/Affirmative Action Institution.

Inductive Learning of Decision Rules
from Attribute-Based Examples:
A Knowledge-Intensive Genetic Algorithm
Approach

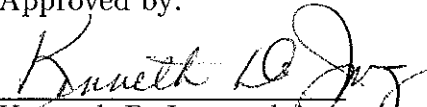
by

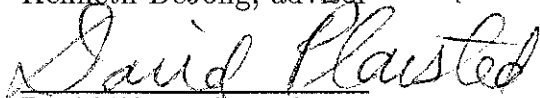
Cezary Z. Janikow

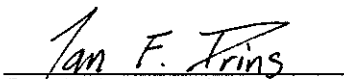
A dissertation submitted to the faculty of the
University of North Carolina at Chapel Hill in partial
fulfillment of the requirements for the degree of Doctor of
Philosophy in the Department of Computer Science.

Chapel Hill, 1991

Approved by:


Kenneth DeJong, advisor


David Plaisted, reader


Jan Prins, reader

©1991
Cezary Z. Janikow
ALL RIGHTS RESERVED

CEZARY Z. JANIKOW. Inductive Learning of Decision Rules from Attribute-Based Examples: A Knowledge-Intensive Genetic Algorithm Approach (Under the direction of Kenneth DeJong)

Abstract

Genetic algorithms are stochastic adaptive systems whose search method models natural genetic inheritance and the Darwinian struggle for survival. Their importance results from the robustness and domain independence of such a search. Robustness is a desirable quality of any search method. In particular, this property has led to many successful genetic algorithm applications involving parameter optimization of unknown, possibly non-smooth and discontinuous functions. Domain independence of the search is also a praised characteristic since it allows for easy applications in different domains. However, it is a potential source of limitations of the method as well.

In this dissertation, we present a modified genetic algorithm designed for the problem of supervised inductive learning in feature-based spaces which utilizes domain dependent task-specific knowledge. Supervised learning is one of the most popular problems studied in machine learning and, consequently, has attracted considerable attention of the genetic algorithm community. Thus far, these efforts have lacked the level of success achieved in parameter optimization. The approach developed here uses the same high level descriptive language that is used in rule-based supervised learning methods. This allows for an easy utilization of inference rules of the well known inductive learning methodology, which replace the traditional domain independent operators. Moreover, a closer relationship between the underlying task and the processing mechanisms provides a setting for an application of more powerful task-specific heuristics.

Initial results indicate that genetic algorithms can be effectively used to process high level concepts and incorporate task-specific knowledge. In this particular case of supervised learning, this new method proves to be competitive to other symbolic systems. Moreover, it is potentially more robust as it provides a powerful framework that uses cooperation among competing solutions and does not assume any prior relationships among attributes.

Preface

The main goal of this project was to show that modeling simple ideas of nature, those utilized in the computational framework known as genetic algorithms, could be successfully applied to supervised inductive learning if such atomic nature-modeled mechanisms were abstracted to the conceptual level of the problem. Such ideas are not completely new. Some researchers previously argued for the incorporation of task-specific knowledge into genetic algorithms applied to machine learning as well as to other domains. However, their approaches could be described as a combination of the traditional approaches with some problem specific operators. Moreover, no such approach was implemented for the particular task of building symbolic concept descriptions in attribute-based spaces. My idea here was just to use the framework of genetic algorithms, and to organize the search by a set of operators designed independently from the underlying mechanisms. Having as much interest in showing the feasibility and potentials of such an approach as in proving grounds for a subsequent development of an actual system, a special emphasis is placed on dealing with some design and implementation issues.

At the same time, I was pursuing two secondary goals. Firstly, I tried to find some common ground between such a genetic algorithm abstraction to the problem level and the more traditional artificial intelligence problem solving paradigm of production systems. The idea was that some of the negative characteristics of genetic algorithms, especially high time complexity, could be further improved by exploring and employing some production system ideas. Secondly, a natural by-product of this work is a special tool allowing for a deeper study of the theory and methodology of symbolic inductive learning since this approach can be viewed as an implementation of such a methodology with the liberal and robust search control of genetic algorithms.

A note on notation: all relevant terms being explained are first shown in *italic*.

I would like to thank all people who directly and indirectly supported me and provided valuable insights. In particular, I would like to thank Ryszard Michalski for an excellent introduction to inductive learning and scholastic attitudes, Zbigniew Ras for his continuous support throughout my graduate years, and Kenneth DeJong with Zbigniew Michalewicz for introducing me to the ideas of genetic algorithms.

Special thanks are reserved for my wife, Grażyna, for her understanding and love.

Contents

1	INTRODUCTION	1
2	GENETIC ALGORITHMS	4
2.1	What Genetic Algorithms Are	4
2.2	Why Genetic Algorithms Work	6
2.3	How Genetic Algorithms Work: an Example	7
3	INDUCTIVE LEARNING FROM EXAMPLES	9
3.1	Problem Statement	9
3.2	Language	10
3.3	Conceptualization	12
3.4	Goals of Knowledge Acquisition	13
3.5	Incremental Learning	14
3.6	Inference Rules of the Inductive Methodology	15
4	PREVIOUS APPROACHES	16
4.1	Traditional Approaches	16
4.1.1	Non-symbolic	16
4.1.2	Symbolic	17
4.2	Genetic Algorithm Approaches	18
5	THE MODIFIED GENETIC ALGORITHM	20
5.1	Ideas Used	20
5.2	Representation and Search Space	24
5.3	Initial Population	24
5.4	Evaluation Mechanism	25
5.5	Operators	26
5.5.1	Definitions	27
5.5.2	Dynamic aspects	35
5.6	Algorithm	37
6	SOME IMPLEMENTATION ISSUES	38
6.1	Sampling Mechanism	38
6.2	Internal Representation	38
6.3	Data Compilation	40

7	A TRACE OF THE SYSTEM'S BEHAVIOR	42
7.1	Data Compilation	43
7.2	Initialization	44
7.3	Initial Evaluation	44
7.4	One Iteration	45
	7.4.1 Selection	46
	7.4.2 Reproduction	46
	7.4.3 Evaluation	47
7.5	The Remaining Iterations	47
8	AN ALTERNATIVE VIEW	51
8.1	Database	54
8.2	Operators	54
8.3	Control	55
8.4	Efficiency Considerations	56
9	EXPERIMENTS	57
9.1	Experimental Methodology	57
9.2	Emerald's Robot World	58
9.3	DNF Concepts	60
9.4	Multiplexers	62
9.5	Breast Cancer	65
9.6	Dealing with Empty and Invalid Rules	66
9.7	Incremental Learning	67
9.8	Initialization	68
10	CONCLUSIONS AND FURTHER RESEARCH	70
10.1	Parameter Abstraction	71
10.2	Multiple Concepts	72
10.3	Noisy Information	74
10.4	Other Operators	74
10.5	More Intelligent Initialization	75
	Bibliography	76

List of Tables

5.1	Completeness and consistency measures.	25
7.1	Application probabilities.	43
9.1	Error rate summary in the robot world.	59
9.2	GIL's error rate in the robot world.	59
9.3	Complexity's summary in the robot world.	60
9.4	GIL's complexity in the robot world.	60
9.5	GIL's accuracy on multiplexer f_{11}	64
9.6	Summary of the breast cancer experiment.	66
9.7	The effects of retaining invalid and empty rules.	67

List of Figures

2.1	A genetic algorithm.	5
2.2	Examples of mutation and one-point crossover.	5
2.3	An example of a genetic algorithm at work.	8
3.1	An example of a single concept learned by one description.	13
3.2	The usage environment for a learning system.	14
5.1	The spectrum of knowledge incorporation in a GA.	21
5.2	Application of a GA with traditional operators.	22
5.3	Application of a task-specific genetic algorithm.	23
5.4	A genetic algorithm with traditional operators.	23
5.5	A genetic algorithm with task-specific operators.	23
6.1	An internal representation of a sample chromosome.	39
6.2	Examples of binary coverage vectors of a feature.	41
7.1	The goal concept and the training events.	44
7.2	The best initial chromosome.	45
8.1	A production system architecture.	52
9.1	Diagrammatic visualization of the acquired knowledge.	61
9.2	Batch-incremental results on DNF data.	63
9.3	Learning curve on multiplexer f_6	64
9.4	A sample behavior on multiplexer f_{11}	65
9.5	Comparison of the batch and incremental learning.	68
9.6	The effect of initial hypotheses in the initial database.	68
10.1	A sample result of learning multi-descriptions.	72
10.2	A sample internal representation of a two-concepts case.	73

Chapter 1

INTRODUCTION

The problem of automatic knowledge acquisition is a central issue in *machine learning*, a subfield of artificial intelligence (AI) devoted toward designing methods and methodologies for both knowledge representation and self-creation or self-reorganization of such knowledge by automata. A big part of that research is devoted toward restricted attribute-based spaces. Such popularity can be attributed to the existence of many practical problems without a sufficiently understood body of knowledge, but with widely available data in the form of feature descriptions.

Traditionally, all approaches to automatic knowledge acquisition have been classified into *non-symbolic* and *symbolic*, depending on the output language. Non-symbolic inductive learning systems, often called *subsymbolic*, usually do not acquire any explicit knowledge but rather gather other information necessary for the descriptive process. They include statistical models, where the only representation is by means of all stored examples or some statistics on them, and the connectionist models, where the knowledge is distributed among a network's connections and an activation method. Symbolic systems, on the other hand, produce explicit knowledge in a high level descriptive language. However, an equally important distinction can be based on the level of inference. All non-symbolic approaches process low level entities (usually numerical parameters). On the other hand, the level of inference of symbolic approaches widely varies, with those operating at higher level showing an advantage: mechanisms based on symbol manipulations, in addition to being closely related to their task-objectives, allow for the use of the same input and output language. This, in turn, creates the possibility of employing some more sophisticated learning paradigms, *e.g.* closed-loop learning ([44]).

With the development of the computer there has been an increasing interest in simulating nature as means for problem solving. One of the most known frameworks was developed by Holland ([22]) and is known as *genetic algorithms* (GAs). This approach models the natural processes of inheritance of coded knowledge and survival by fitness or degree of adaptation to the environment. The two most important characteristics of GAs are robustness and domain independence of their search mechanism. Robustness, an ultimate goal of any system, is a natural by-product of the search strategy which performs simultaneous space *exploration* and *exploitation* (global and local search). This makes the mechanism quite independent of the characteristics

that normally break most other approaches, such as non-smoothness or discontinuity of the evaluation function. Domain independence, on the other hand, is obtained by designing the search operators in the space of the lowest level representation entities. However, such an approach has both advantages and disadvantages. On the positive side is the fact that a new application requires only a coding of the problem to this artificial space. On the negative side lies the fact that the quality of such a coding is crucial to the genetic algorithm's performance. Moreover, operating in this space means using problem-blind operators, which often overlook some important information that normally could be utilized to guide the search.

Nevertheless, genetic algorithms have been quite successfully applied to a number of problems. The most outstanding results come from the field of parameter optimization (*e.g.* [9, 12]), where the mentioned coding is rather easy and straightforward. Moreover, a non-standard floating-point representation seems to provide for additional effectiveness (*e.g.* [34, 35]). Other successful applications include optimization problems like wire routing and scheduling, game playing, cognitive modeling, transportation problems, the traveling salesman problem, and optimal control problems (*e.g.* [5, 8, 10, 21, 18, 63]). However, applications to machine learning, although partially successful ([32, 53, 57, 58]), never succeeded in more complex domains and encountered many additional problems ([12]). The genetic algorithm approach to supervised learning in an attribute-based space is normally referred to as symbolic. However, the processing is normally done in symbols of the artificial, not the problem, language (with some exceptions, *e.g.* [32]). This mismatch is the main reason for the low rate of success. Recognizing this, there have recently been a few attempts to find a more problem related representation ([13, 19, 58, 61]), but they generally still fail to provide symbol processing at an appropriately high conceptual level.

We propose to use a *rule-based* representation (with condition-action pairs) as the natural choice for a symbolic system operating in this space. Having that, we propose to use operators that directly model the inference rules defined in such a framework, namely those of the inductive learning methodology (described by Michalski in [39]). By doing this, we utilize the task-specific problem solving methodology and abstract the genetic algorithm's inference to the problem-specific symbol level. This can be viewed as a *knowledge-intensive* approach, *i.e.* using a vast amount of task-specific information, which replaces the blind search of traditional domain-independent operators by a heuristic search. Implementing all the extra knowledge in the operators leaves the remainder of the genetic algorithm intact and allows for an easy extension of these ideas to other domains. Because of the richness of such new operators and their domain-dependent behavior, the new algorithm does not enjoy the same theoretical foundations as the traditional GAs do. Nevertheless, we try to show how to justify it intuitively, and the results of our experiments show its applicability.

This approach can be seen as a genetic algorithm for processing high level structures specific to the problem. Because of this change, one may question whether it is still a genetic algorithm. We do not attempt to deal here with this delicate issue since it would require a clear definition of the boundaries of genetic algorithms. However, such boundaries are not well defined and are problematic on their own.

The same ideas can be also derived from the artificial intelligence and machine

learning point of view. Then, this approach can be seen as a modified production system that uses stochastically firing task-specific inference rules directly on formulas of a rule-based framework. To deal with the problems of weak heuristics and huge search spaces, the very liberal and robust control mechanism of genetic algorithms is used. This leads to a potentially very robust design which does not assume any prior relationship among different attributes (as, for example, ID does). The robustness is a result of the existence of the platform for both cooperation (by information exchange) and competition among many different simultaneous solutions. This, in turn, can be seen as an extension of the AQ's ideas of processing competing directions (AQ does it by retaining a number of partial covers simultaneously, as explained in chapter 3). Here, we provide the cooperation and use more powerful heuristics (the inference rules and their adjusting applicabilities).

This dissertation is organized as follows. In chapter 2 we describe genetic algorithms and present their theoretical foundations along with some intuition behind their applicability. In chapter 3 we define the problem of supervised learning of concept descriptions from feature-based examples, describe some important related issues, and outline the well-known inductive learning methodology. In chapter 4 we briefly describe the most known approaches to supervised learning, including traditional symbolic and non-symbolic methods as well as those based on genetic algorithms. In chapter 5 we describe the ideas leading to this new approach, followed by a detailed description of the resulting algorithm's components. In chapter 6 we try to deal with some important implementation issues, and in chapter 7 we further illustrate the system's behavior by tracing a sample application. Then, in chapter 8 we try to justify and describe the same approach from the point of view of artificial intelligence and machine learning. We follow with a systematic experimentation in chapter 9. Finally, in chapter 10 we draw some conclusions and describe work to be done in the future in order to produce a complete learning system.

Chapter 2

GENETIC ALGORITHMS

In this chapter we introduce the idea of genetic algorithms, present some theoretical foundations behind their applicability, and conclude with a simple example.

2.1 What Genetic Algorithms Are

Genetic algorithms are adaptive methods of searching a solution space by applying operators modeled after the natural genetic inheritance and simulating the Darwinian struggle for survival. They belong to the class of probabilistic algorithms, yet are distinguished by their different search method and relative insensitivity to local traps.

In general, a GA performs a multi-directional search, and it encourages information formation and exchange between such directions. It does so by maintaining a population of proposed solutions (*chromosomes*) for a given problem. Each solution is represented in a fixed alphabet (usually binary) with an established meaning. The population undergoes a *simulated evolution*: relatively “good” solutions produce offspring, which subsequently replace the “worse” ones. The estimate of the quality of a solution is based on an evaluation function, which plays the role of an environment. The existence of such a population provides for the superiority of genetic algorithms over pure *hill-climbing* methods (*i.e.* methods that seek a goal by always following the direction of the best outlook), for at any time the GA provides for both exploitation of the most promising solutions and exploration of the search space.

Each iteration, called a *reproduction cycle*, is performed in three steps (see figure 2.1). During the *selection* step a new population is formed from stochastically best samples (with replacement). Then, during the *recombination* step some of the members of the newly selected population are altered. Finally, all such altered individuals are evaluated.

The mating process (recombination) itself is based on the application of two operators: *mutation* and *crossover*. Mutation introduces random variability into the population, and crossover exchanges random pieces of two chromosomes in the hope of propagating partial solutions (see figure 2.2). Because both of these operators are defined on syntactic pieces of the underlying representation (when each chromosome is viewed as a sequence of the symbols of the low level alphabet), the search has

```

procedure genetic algorithm
begin
   $t = 0$ 
  initialize  $P(t)$ 
  evaluate  $P(t)$ 
  while (not termination-condition) do
    begin
       $t = t + 1$ 
      select  $P(t)$  from  $P(t - 1)$ 
      recombine  $P(t)$ 
      evaluate  $P(t)$ 
    end
  end

```

Figure 2.1: A genetic algorithm.

domain-independent properties. However, the applicability of a GA to a particular problem depends on the representation emphasizing meaningful semantic pieces of information (called *building blocks*) to be used by the crossover operator, in addition to the evaluation function properly guiding the search. Then, the applicability is often decreased if the operators are defined to manipulate lower level syntactic structures — as is often the case.

<p>mutation: $x_i^t = \langle b_1 \dots b_k \dots b_n \rangle$ \hookrightarrow $x_i^{t+1} = \langle b_1 \dots \bar{b}_k \dots b_n \rangle$</p>	<p>crossover: $x_i^t = \langle b_1 \dots b_k b_{k+1} \dots b_n \rangle$ $x_j^t = \langle d_1 \dots d_k d_{k+1} \dots d_n \rangle$ \hookrightarrow $x_i^{t+1} = \langle b_1 \dots b_k d_{k+1} \dots d_n \rangle$ $x_j^{t+1} = \langle d_1 \dots d_k b_{k+1} \dots b_n \rangle$</p>
---	--

Figure 2.2: Examples of mutation and one-point crossover.

Specifying a genetic algorithm for a particular problem involves describing a number of components. Among them, the most important are:

- A genetic representation for potential solutions to the problem, which also defines the search space of the algorithm.
- A method of generating the initial population of potential solutions.
- An evaluation function that plays the role of the environment, rating solutions in terms of their “fitness” or “adaptation” to this environment.
- Genetic operators that alter the composition of chromosomes during recombination.
- Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, *etc.*).

2.2 Why Genetic Algorithms Work

The theoretical foundations of genetic algorithms rely on the notion of a *schema* (e.g. [22]) — a similarity template allowing an exploration of similarities among chromosomes. A schema is built by introducing a new *don't care* symbol (\star) into the alphabet of genes — such a schema represents all strings (a hyperplane, or subset of the search space) that it matches on all positions other than \star . Assuming a binary alphabet and a population of size n , there are between 2^n and $n \cdot 2^n$ different schemata represented; at least n^3 of them are processed at any time — Holland has called this property an “implicit parallelism”, as it is obtained without any extra memory/processing requirements.

Two other important notions associated with the schema are necessary to derive the theoretical basis:

- *Schema order*, $o(H)$, is the number of *defining*, i.e. non don't care, positions. Essentially, it defines the speciality of a schema.
- *Schema defining length*, $\ell(H)$, is the distance between the first and the last defining symbols of a chromosome. It defines the compactness of information contained in a schema.

Assuming that the reproductive probability is proportional to fitness, we can derive the following two versions of the growth equation (e.g. [21]):

$$m(H, t + 1) = m(H, t) \cdot \frac{f(H, t)}{\overline{f(t)}}$$

where $m(H, t)$ is the number of schemata H at time t , $f(H, t)$ is the average fitness of schemata H at time t , and $\overline{f(t)}$ is the average fitness of the population, and the recursive version

$$m(H, t) = m(H, 0) \cdot (1 + c)^t$$

where c is the above-average part of H 's fitness.

These two equations show that the selection increases sampling rates of the above-average schemata and that this change is exponential. However, no new schemata (not represented in the initial $t = 0$ sampling) can be formed, which prohibits the application of the selection alone. This is exactly why the recombination phase is introduced: crossover enables structured, yet random, information exchange, and mutation introduces additional variability into the population. Therefore, to formulate the complete theory, we must consider the effect of both of these operators on the growth equation. Assuming independent probabilities p_c and p_m for one-point crossover and mutation respectively, we obtain:

$$m(H, t + 1) \geq m(H, t) \cdot \frac{f(H, t)}{\overline{f(t)}} \cdot \left[1 - p_c \frac{\ell(H)}{l - 1} - p_m \cdot o(H) \right]$$

This result can be stated as ([21]):

Schemata Theorem: *Short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.*

An immediate result of this theorem is that GA explore the search space by short schemata which subsequently are used for information exchange during crossover:

Building Block Hypothesis: *A genetic algorithm seeks near optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks.*

Although some effort has been done to prove this hypothesis (e.g. [4]), for most nontrivial applications we rely on empirical results. Nevertheless, this hypothesis suggests that the coding problem for a genetic algorithm is critical for its performance, and that such a coding should emphasize meaningful building blocks. This, in turn, suggests the following intuitive approach to problem solving by genetic algorithms:

Intuition: *The problem representation in a GA should be such that conceptually related alleles are close together in the resulting genotype.*

2.3 How Genetic Algorithms Work: an Example

We close this chapter with an illustration of a genetic algorithm at work (figure 2.3). Consider a multimodal function of one variable $f(x)$, with a given domain $x \in (a, b)$. Using the binary alphabet, we code this variable as a chromosome. The number of bits in such a representation depends on the variable's domain ($b - a$) and a desired precision. Assuming ten binary bits, a chromosome (and, therefore, a potential solution) may look as follows:

0011101011

Of course, to evaluate the fitness of each such binary number, it first must be translated to its decimal equivalence (235 here) and then scaled to the domain (a, b) . For example, the above chromosome translates to the value:

$$evaluation = 235 \cdot \frac{b-a}{2^{10}-1}$$

Then, we decide the population size (twenty in this case), and fill it with randomly generated chromosomes ($t = 0$). This random initialization corresponds to sampling the search space. We evaluate the samples (the dots in the reference figure represent the evaluations of the chromosomes). Next, we select a new population with higher-valued chromosomes having proportionally bigger probability of appearance, and we apply the genetic operators to produce new samples. Here are two examples of possible mutation and crossover:

mutation:	crossover:
0011101011	0011101011
↔	1100000010
0010101011	↔
	0000000010
	1111101011

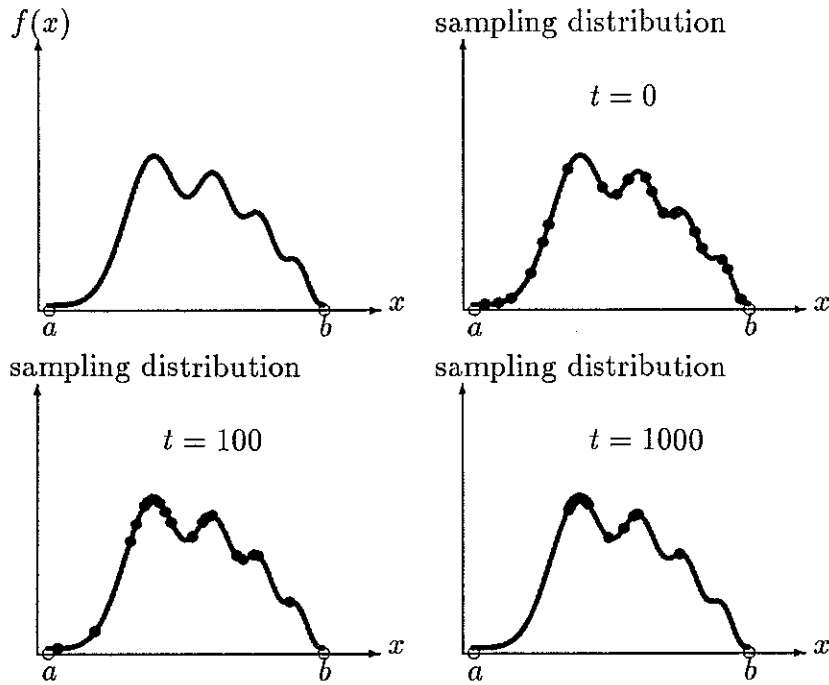


Figure 2.3: An example of a genetic algorithm at work.

We iterate the evaluate/select/recombine process a number of times. After a number of such iterations, the sampling concentrates around the high-payoff subspace (see figure 2.3 after 100 and 1000 iterations). With a sufficient number of such iterations the solution can be found (or closely approximated, in general).

Chapter 3

INDUCTIVE LEARNING FROM EXAMPLES

In this section we define the problem of inductive learning from examples of an attribute-based space and describe some important related issues. We also present inference methods of the well known inductive learning methodology, which we subsequently use as the basis for our new approach.

3.1 Problem Statement

Concept learning is a fundamental cognitive process that involves learning descriptions of some *categories* (*i.e.* sets drawn from a common class) of objects. Such descriptions can be placed in different universes. We are considering here the restricted *attribute-based spaces*, spanned by a number of attributes, each of which has a finite set of allowed values (in essence, this is a generalization of boolean functions from two- to multi-valued domains). A priori knowledge consists of a set of *events*, *i.e.* examples of the space. Each such an event is actually a point in the attribute-based problem space. Moreover, we are considering here the case of *supervised* learning, *i.e.* learning assuming that each a priori event is preclassified as an example of one of the concepts to be learned.

In other words, each concept is described in terms of a set of sample events. Its members are called *positive examples* of the concept. The task is to generalize the a priori knowledge in order to produce descriptions of such concepts. When a *rule-based* framework is used to express such descriptions, the acquired knowledge is often called *decision rules*. Such rules can subsequently be used to both infer properties of the corresponding classes and to classify other, previously unclassified, events from the space. For example, consider the case of the cardiac unit's database of a hospital, where each patient is described in terms of a fixed number of attributes including his/her personal information, clinical history, and results of various tests. Assuming that all numerical measurements are divided into discrete intervals, each patient becomes a single event in the finite-sized space. Furthermore, assume that some of the patients are known to have a certain heart disease, while others are known

not to have it. Finding a feature-spanned description of the first group of patients certainly would help us define groups of people in high risk groups, as well as help us predict the risk of developing this disease in new patients.

The generated knowledge can, following various intentions and criteria, describe *characteristic* or *discriminant* properties of the categories ([39]). Characteristic are the most typical properties, and should be maximal, *i.e.* listing a maximal number of appropriate *features* (a feature is an attribute-value pair). This kind of learning is often conducted on positive events only. Discriminant are the properties necessary to differentiate a given concept from others, and normally such a minimal set is sought — this learning requires both positive and *negative* events (negative events are those representing other classes). In order for the sets of positive and negative events to be disjoint, the descriptive power of the features must be sufficient. Such a scenario (called *consistent*) is desired, but not necessary for the learning (if the initial data set is inconsistent, a special protocol for treatment of such problematic examples must be employed).

3.2 Language

An important issue is that of defining both the input and the output language. The input language serves as an interface between the environment (as a teacher) and the system. Therefore, it should combine requirements of both these entities. Moreover, it should minimize inconsistencies among data. The output language serves as an interface between the system and the application environment. Therefore, it should combine the requirements of the learning system with those of such an environment. For example, for a purely classification application, there is no need to express the acquired knowledge on any comprehensive level. The output interface is only to provide obtained classifications of some new events. On the other hand, a learning agent used as a part of a hybrid intelligent system must be able to communicate its knowledge to other parts of the whole system (see figure 3.2). If such a system contains elements operating in a high level language (as an expert system, human expert, *etc.*), our learning agent should be able to express its knowledge at the same level.

Learning systems unable to produce high level knowledge are often classified as *non-symbolic*, and the other ones as *symbolic*. However, such a classification does not reflect the important issue of the language of the inference mechanism; in [69] the authors call systems that produce high level output but operate on a lower level *heterogeneous*. We try to make a similar distinction where we feel it is important.

In the non-symbolic systems the output knowledge usually consists of a set of numerical parameters. In the symbolic systems two choices proved to be dominating: decision trees ([49]) and decision rules ([38]). Rules normally have the advantage of being the same on both the input and the output, which facilitates some important learning strategies as *incremental* or closed-loop learning. However, this advantage is rather an elegance, since it has been shown that decision trees can be both converted to rules ([51]) and applied in an incremental environment ([62]). Nevertheless, the

difference actually goes beyond the elegance if one considers the level of inference: the rule-based systems normally also operate on structures of the same language.

One widely used language, which is not only closely associated with rules, but is normally also used to represent input events for any program operating in an attribute-based space, is VL_1 ([41]). *Variables* (attributes) are the basic units having multi-valued domains. According to the relationship among different domain values, such domains may be of different types:

- *Nominal*, e.g. { *Yes*, *No* } in boolean attributes
- *Linear*, e.g. *AgeOfPatient*, with linearly ordered values
- *Structured*, with partially ordered values

Relations “=, ≠, <, ≤, >, ≥” associate variables with their values by means of *selectors* having the form [*variable relation value*], with the natural semantics. For example, [*Age > Young*] is interpreted as the set of people of *Middle* or *Old* age, assuming the three values (*Young*, *Middle*, *Old*) in the domain of the linear attribute *Age*. Notice that while the “=, ≠” relations can appear in any selector, the other four can only be used with ordered domains. The value in a selector is a single domain value. However, for the ‘=’ relation it may be a disjunction of such values. In this case we call it an *internal disjunction*: this greatly reduces the complexity of the formulas of the language. Conjunctions of selectors form *complexes*.

The significance of the language relies on the natural correspondence to a rule-based paradigm. Selectors can be used to express conditions on single attributes. Complexes can be used to express rules of the form:

complex ::> *decision*

Because of that, the semantics of the language’s constructs are easily understood. For example, the following set of rules describes people with heart problems as those which are older and have high blood pressure, or those with high cholesterol levels:

[*BloodPressure = High*] [*Age ≠ Young*] ::> *HeartRiskGroup*
 [*CholesterolLevel = High*] ::> *HeartRiskGroup*

The internal disjunction, along with the natural semantics, makes the equality relation alone sufficient to represent any formula of the language. For example,

- Assuming that the attribute *A* has a linearly ordered domain:

$$[A \geq v] \iff [A = v, v_i, \dots, v_k]$$

where v_i, \dots, v_k are all domain values greater than v in the ordering.

- For any attribute *B*:

$$[B \neq v] \iff [B = v_1, \dots, v_k]$$

where v_1, \dots, v_k are all domain values different from v .

This fact provides for great simplicity and uniformity to further discussion of processing the elements of the language, as it allows us to define all the operators in terms of the equality relation alone (chapter 5 and 6).

3.3 Conceptualization

The idea of a concept itself may be defined in many different ways, depending on the assumed concept representation and on methods of instance classification. According to Smith and Medlin ([60]), historically there have been three basic approaches:

- The *classical view*, which assumes that all instances of a concept share some common properties which uniquely identify the concept. It assumes that concepts are complete and consistent descriptions of categories.
- The *probabilistic view*, which states that the common properties are the most typical, but any instance does not have to possess all of them. The concept membership is decided by an accumulated degree of fitness.
- The *exemplar view*, where a concept is represented by its most typical examples (*exemplars*), and the concept membership is decided by a measure of fit to an exemplar.

Each of these views has been criticized for its inability to precisely model human conceptualization. For example, the classical view has been criticized for its inability to represent disjunctive and imprecise concepts. The first of these disadvantages is relaxed by a rule-based view. However, such a view is *crisp* as it does not account for imprecise concepts. The exemplar and probabilistic views, on the other hand, were mostly criticized for ignoring the role of generalization. The most recent trends seem to explore hybrid views, combining some crisp approaches (as rules) with probabilistic ones ([15]) as means of dealing with noise and imprecise concepts. For example, the *two-tiered* approach employed in AQ15 combines a rule-based view representing most typical properties with an inferential quantitative part accommodating boundary and imprecise cases ([42]).

In addition to psychological evidence, the choice of the conceptualization method is often dictated by the underlying knowledge acquisition method, or rather its output language. For example, assuming a sufficient set of attributes for the events to be consistent and a crisp view, a decision tree can naturally produce *complete* and *consistent* partition of the search space (a search space is covered completely if every subspace is covered, and it is covered consistently if no subspace is covered in a conflicting manner). This is so because the decision tree mechanism starts with the whole space and recursively cuts it into disjoint subspaces. On the other hand, it is more difficult for a set of rules to cover the search space in the same manner (normally such qualities are required only with respect to the set of training events; in such a case we say that the rules are complete/consistent with respect to the training

data set). Therefore, an extra mechanism is needed to account for possible cases of no-match and multiple-match. Such problems can be avoided while learning simple concepts (called *single-concepts*) if the system learns only the concept description and its negation is assumed to represent the negation of the concept (see figure 3.1).

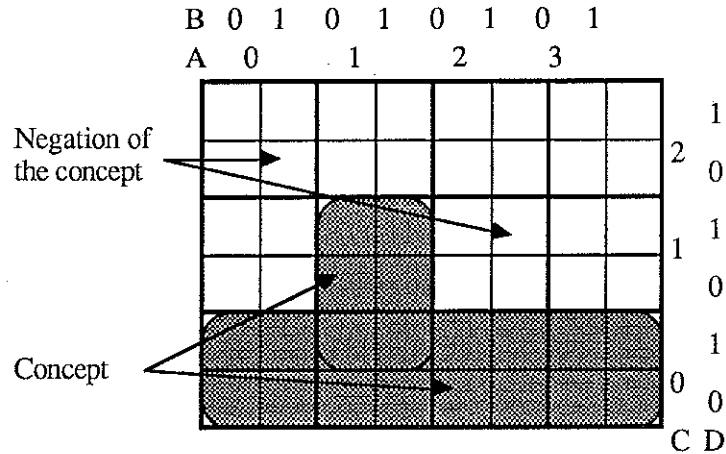


Figure 3.1: An example of a single concept learned by one description.

3.4 Goals of Knowledge Acquisition

The purpose of extracting information about some concepts of an attribute-based space is to acquire knowledge able to:

- Predict classifications of previously unseen examples by assigning confidence factors to different hypotheses.
- Interact with other entities of a hybrid intelligent system, *e.g.* with a human or an artificial expert. For ease and feasibility of such interactions, the language of the system should be coherent with that of the other entities.

Until now, most automatic acquisition systems were used as either direct classification systems, or their output was to be further processed by hand. This is why most experiments evaluating qualities of such systems concentrated on the prediction accuracy (see *e.g.* [25]). Nevertheless, there were sporadic attempts to address the other issue as well (*e.g.* [54, 69]). It seems appropriate to say that while the prediction accuracy will continue to have high importance in future learning systems, the ability to express the knowledge on a high abstract level will play an increasingly important role. An additional reason for that, pointed out by Michalski in [43], relates to the increasing dependence on any automatically generated knowledge:

“An important implication . . . is that any new knowledge generated by machines should be subjected to close *human scrutiny* before it is used.

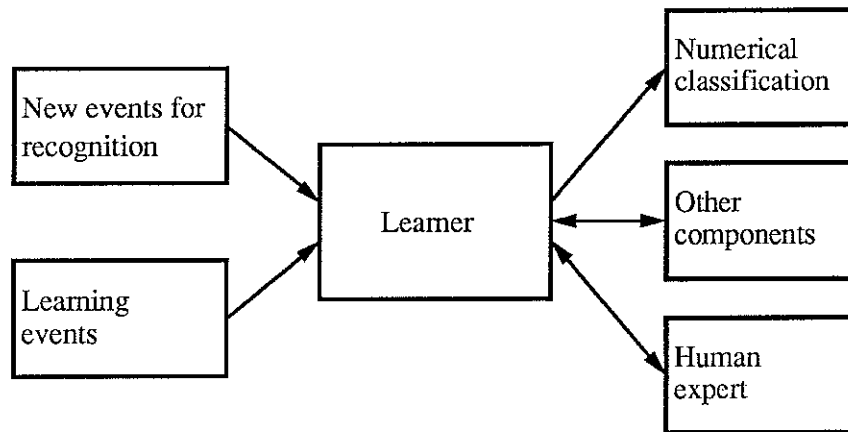


Figure 3.2: The usage environment for a learning system.

This suggests an important goal for research in machine learning: if people have to understand and validate machine-generated knowledge, then machine learning systems should be equipped with adequate *explanation capabilities*. Furthermore, knowledge created by machines should be expressed in forms closely corresponding to human descriptions and mental models of this knowledge; that is, such knowledge should satisfy what this author calls the *comprehensibility principle*.”

3.5 Incremental Learning

According to Holland ([23]), an inductive process in a rule-based framework must accommodate both revision of rules and generation of new ones. Generation of new rules is associated with two sources, the environment and existing knowledge. According to Michalski ([39]), an inductive learning system must accommodate the production of new rules, as well as generalization and specialization of existing ones. According to Winston ([67]), near-miss examples, carefully chosen by the teacher, specialize existing models, while correct examples further generalize them. All the above, in addition to some psychological evidence, support the idea of incremental concept learning. First concept descriptions are formed early, after seeing few examples. Upon experiencing new evidence, the knowledge is revised:

- If the new evidence is consistent, the knowledge might be further generalized, as the confidence in certain assertions increases.
- If the new evidence is inconsistent, the knowledge must be refined by specialization of the overgeneralized assertions.

In addition, normally not all examples of a concept are available simultaneously. This is true for both natural and artificial systems. Therefore, a learning agent should be able to perform incremental conceptualization, unless special circumstances exist (such as a relatively static world), which permit a *batch* learning (with the assumption

that all available examples are available simultaneously). Accordingly, incremental capabilities are receiving increasing attention of the research community ([15]).

There are two different approaches to learning in the context of increasing number of examples. The first assumes the existence of working memory able to remember all previously seen events for a future reference. This approach is normally referred to as *full memory learning*, and the incremental processing is associated with both previously generated knowledge and previously seen examples, in addition to the newly presented events (one should also mention here the case of so called *batch-incremental* systems, which process the incrementally available events in relation to only the previously seen events, disregarding the previously generated knowledge). The feasibility of such full memory systems is restricted by the data set size. However, its other advantages, especially relative conceptual simplicity, cause many learning systems to follow this direction. In addition, two other important factors favor this approach in the domain of attribute-based spaces: the available data sets for many interesting concepts are appropriately small, and such systems can normally accommodate the large data sets by means of some special techniques (see section 4.1.2).

The other approach assumes that the only available memory is for the generated body of knowledge. In general, systems of this kind are conceptually more difficult to apprehend. They find more use in other machine learning subfields, especially where single events are of high complexity.

3.6 Inference Rules of the Inductive Methodology

Michalski ([39]) provides a detailed description of various inductive operators that constitute the process of inductive inference. In the restricted language VL_1 (for induction in an attribute-based space), the most important are: *condition dropping* — *i.e.* dropping a selector from the concept description; *adding alternative rule* and *dropping a rule* — adding/removing one rule from the description; *extending a reference* — extending an internal disjunction; *closing an interval* — for *linear* domains filling up missing values between two present values in a selector; *climbing generalization* — for *structured* domains climbing the generalization tree; *turning a conjunction to a disjunction*; *inductive resolution* — analogous to the resolution principle. These operators are either generalizing or specializing existing knowledge. There is no provision for generating the initial set of rules. In section 5.5 we define our operators. We also discuss the choice of the initial knowledge (section 5.3).

Chapter 4

PREVIOUS APPROACHES

Over the past few decades there have been many different approaches to the problem of supervised learning in attribute-based spaces. Some of them came from the AI community, others from fields such as statistics. One of the most recent ideas has been to use genetic algorithms, to which we pay special attention since our new approach tries to extend such ideas by those of the inductive methodology.

4.1 Traditional Approaches

As mentioned earlier, non-symbolic systems rely mostly on quantitative information processing. Therefore, they are further away from mainstream AI devoted to symbol processing. On the other hand, the symbolic systems apply the qualitative approach to learning: the output is a high level description, and the processing itself is often done at the symbol level.

4.1.1 Non-symbolic

Statistical approaches account for the vast majority of non-symbolic, or numerical, approaches. They usually operate in batch mode on the data set in order to obtain some statistical measures, which are later used as probabilistic approximations of appearances of different features. To achieve a suitable classification, this information's correlation to a new example is accumulated using some inference methods. Among such methods, the Bayesian probabilistic model is the most known ([47]). The disadvantage of these approaches is that they rely on low level processing for high level learning. Furthermore, such treatment makes it hard to process any available problem specific knowledge. In addition, the measures used treat all features independently, and high processing complexity does not allow exploration of inter-feature dependencies ([47]), even though they could be incorporated ([54]).

Another numerical approach comes from the neural network community. A neural network is a cognitive model of the human brain and is composed of two kinds of elements: processing elements (nodes of the network) and connections ([55]). Viewed as a memory, such a network has its knowledge distributed among the connections — called weights. These weights determine the propagation of excitatory and inhibitory

signals which in turn determine the excitation of certain nodes. Such a memory model is capable of learning. The backward propagation of a failure is the best known method of setting the weights. A neural network method has been applied to simple cases of concept learning with some success (see [68]). However, these applications are totally quantitative as well, which makes it difficult to establish a platform for any higher level knowledge utilization or understanding.

As described, the non-symbolic systems do not follow the methods of the inductive learning methodology, but rather perform numerical computations. This, however, leads to an apparent advantage of better applicability to processing noisy information ([54]).

4.1.2 Symbolic

The two prominent symbolic approaches to supervised feature-based learning, recognized as benchmarks, are Michalski's AQ ([39]) and Quinlan's ID ([48]). They are both considered symbolic systems, even though they have some numerical elements: ID uses an information measure function, while the *two-tiered* representation of AQ performs a partially probabilistic inference.

The AQ approach is based on inductive generalization and specialization of the VL_1 formulas using the idea of a *cover* of the positive against the negative events. The cover is constructed in an iterative manner, starting with only one positive and one negative event and continuing until the generated cover is complete and consistent. To prevent an apparent exponential growth in the number of generated descriptions, special heuristics, which accommodate some learning criteria, are employed to reduce the size of partial covers. However, retaining a number of such current covers provides for a competition among different solutions. This approach conceptually follows the ideas of inductive methodology, as the generated knowledge is either generalized or specialized, as appropriate. However, the algorithm itself uses only the logic-based operators of negation, union, and intersection to process the current description.

The many proposed extensions of this basic algorithm facilitate incremental learning, constructive learning, use of initial hypotheses and domain assertions, *etc.* (see [40]). In addition, both the most representative examples selection method ([37]) and the two-tiered concept representation ([42]) allow for processing of noisy data.

In the ID approach, the training examples are represented by feature vectors similar to events in VL_1 . The algorithm constructs a decision tree, where each leaf is associated with a single decision class, each internal node corresponds to an attribute, while each node's branches correspond to a value of that attribute. One of the features of such a tree is that no path from the root to a leaf has two nodes corresponding to the same attribute. The algorithm itself is an iterative application of the information content formula: $I = p * \log(p)$, where p is a probability of given information ([48]). At each node of the tree the algorithm only treats events satisfied by the path to this node: the information content is calculated for all such remaining attributes and events, and the attribute giving the maximal information gain is selected as the label for this node.

This approach, despite its apparent disadvantage of "no look-ahead" (a node is

constructed based on the currently best attribute), proved to be successful in terms of recognition quality. However, the numerical formula used causes serious problems while trying to incorporate some domain specific knowledge. On the other hand, recent extensions allow for incremental concept learning ([62]), for reducing the tree to binary ([31]), for pruning the tree, and for converting it to a set of rules ([51]). However, the algorithm is conceptually very distant from the inductive learning methodology. It iteratively applies specialization, starting with the whole event space. Only then generalization can be applied, by means of tree pruning or rule construction techniques.

Both of the above are full memory systems, meaning that they assume the availability of all previously seen events at any time during incremental learning. However, they both allow for processing large quantities of data: the AQ uses a preprocessing mechanism selecting only the most representative events ([37]), and the ID uses the idea of data windowing.

4.2 Genetic Algorithm Approaches

Since the early 1980's there has been an increasing interest in applying GA methods to machine learning — in particular to learning production rules, whose special case is the problem of supervised learning from examples of an attribute-based space. The main problem in such applications is to find a suitable representation, able to both capture the desired problem characteristics and to represent a potential solution (as we mentioned in section 2.2). Using a rule-based concept representation brings a different kind of problem: the number of such rules (disjuncts) is not known a priori. Therefore, the traditional fixed length representation is unsuitable. Two different approaches have been proposed:

- **Michigan** approach, where the population still consists of fixed length elements, but the solution is represented by a set of chromosomes from the population. This methodology, known as CS for classifier systems, along with a special “bucket brigade” mechanism for credit assignment, was originally developed by Holland and colleagues ([23]). Here, each chromosome, called a classifier, represents a structure composed of conditions and messages lists. The environment, together with the activated rules, provides a set of active messages. These, in turn, activate other classifiers by satisfying their conditions. The chained actions of message-condition pairs cluster the rules together. Because of this chaining mechanism, this approach seems more suitable for planning than concept learning.
- **Pitt** approach, which represents an extension of the traditional fixed-length chromosome approaches. Here, variable length chromosomes are used to represent proposed solutions individually. Such a representation (LS for Learning System) was originally suggested and theorized by Smith in [61]. This representation seems more naturally suited for the supervised learning from feature-based examples problem.

Both of these approaches suffer from some drawbacks: the classifier systems rely on the problematic bucket brigade for a nontrivial credit assignment. The variable length approach constitutes a wide divergence from the traditional GA, and, therefore, requires special treatment. Nevertheless, some applications prove to be successful, although in quite limited applications (*e.g.* [19, 57, 66]). The two most noticeable genetic algorithm approaches to supervised concept learning in attribute-based spaces come from Koza and DeJong with Spears, both in the LS framework. Koza uses lisp programs (similar to decision trees) as means of representing potential solutions, with some tree-based operators that are closed in the space of such representation: some results are presented in [32]. DeJong and Spears use a binary representation for multi-valued domains, and implement only the traditional operators of mutation and crossover in their GABIL system. In [13] they compare GABIL's batch-incremental learning with those of ID5 ([62]) in the domain of random DNF descriptions.

With very few exceptions (*e.g.* [13, 32]), all systems for rule-based learning use a three-symbol alphabet $\{0, 1, \#\}$, where $\#$ stands for a wild-card character (*e.g.* [19, 17, 57]). Such an alphabet is the choice of both CS and LS based systems, as it allows for easy coding of various general aspects of machine learning. However, it is not so well suited for non-binary domains, where an attribute can take a value from an unpredictably (but assumed finite) sized domain: in particular it is not well suited for feature-based spaces. This problem was originally pointed out by Greene and Smith in [17], and a nice solution was employed by DeJong and Spears in the previously mentioned publication ([13]).

Chapter 5

THE MODIFIED GENETIC ALGORITHM

In this chapter we first describe the major ideas used to derive the system's design, and then present the system itself by defining the necessary components (those listed in section 2.1). Some implementation issues, strongly associated with any such approach, are addressed in the next chapter.

5.1 Ideas Used

One of the most praised characteristics of a genetic algorithm is its domain-independent search ([12]). This is the source of both the many successes of GAs, especially in parameter optimization, and, at the same time, of many limitations in other applications (see chapter 1). In general, such arguments fall into the same category as those calling for more domain specific AI methods two decades ago. Recall that, at first, general problem solvers (GPS) were devised — which were to play the role of general tools for many nontrivial problems. It soon turned out that, due mostly to the unmanageable complexity of such methods, it was necessary for the designers of intelligent systems to incorporate domain and problem specific knowledge, by either making it explicit or by hiding it in the implementation.

The need for problem specific knowledge incorporation into GAs was recognized as a method for an improvement in many different domains (*e.g.* [18]). Davis calls for such approaches in [8], where he calls them “hybrid” genetic algorithms and argues to explore combinations of GAs with existing methodologies in any possible domain. Similar ideas were called for in applications to machine learning. For example, Forrest proposed to use high level operators ([16]) such as “concept specialization” and “value restriction”; Antonisse and Keller called for similar incorporations in [1]. However, both of the above were restricted to classifier systems and the three-symbol alphabet.

Following the Pitt approach, which seems more natural for our task of learning disjunctive descriptions, we are faced with chromosomes of varying number of fixed length structures. Then, there is a whole spectrum of possible GA designs along the dimension of task-specific knowledge utilization (see figure 5.1). On one side

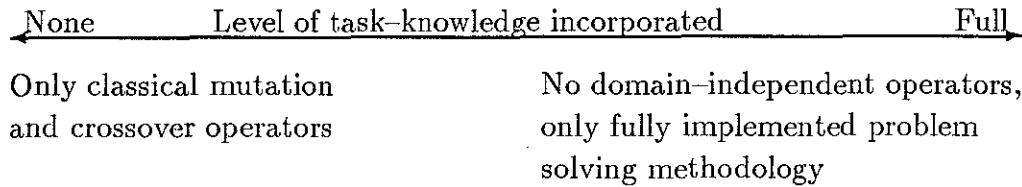


Figure 5.1: The spectrum of knowledge incorporation in a GA.

of the spectrum lies a method which only uses the classical operators of mutation and crossover. Such an approach is conceptually very easy. Moreover, it enjoys the same theoretical foundations as the fixed-length GAs: Smith showed that a variable-length list of fixed-length structures satisfies the Schemata Theorem, provided that such structures are positionally independent ([61]). The above determines the existing popularity of such approaches in any domain. The same is true in the particular case of supervised inductive learning. For example, the previously mentioned GABIL system follow this path.

On the other side of the spectrum lies a knowledge-intensive method which completely abandons the traditional domain-independent operators, and, instead, fully implements the specific problem solving methodology. This approach is conceptually much more challenging as it requires, in addition to the GA implementation, a clear and conscious understanding of the problem being solved, along with a well described complete solving methodology defined at the problem level. This fact, in addition to the lack of well established theoretical foundations, determines the low popularity of such approaches. In particular, we are not aware of any GA for supervised learning in attribute-based spaces working at this end of the spectrum.

From the above discussion a clear trade-off emerges between the two extreme approaches along the spectrum. Domain-independent operators are conceptually easy, while the design of task-specific operators requires deep understanding of the problem solving methodology. The former is also backed by the Schemata Theorem, but the search is problem-blind (except for the measure of fitness, see figure 5.2), and it may easily fail under the restriction of resource limitations (*e.g.* time constraints). The latter approach does not have the same theoretical support, but it is backed by the task-specific knowledge used to guide and conduct the search (see figure 5.3). This property should provide for a faster convergence to a desired solution. Moreover, it may be easily shown that all the operators we subsequently define and use (section 5.5) are actually special cases of the traditional mutation and crossover. This provides an intuitive support for the Schemata Theorem. Also, because the operators are defined on the semantic pieces of the problem, one may easily argue that this design naturally satisfies the building block hypothesis as well (chapter 2).

These are some of the reasons for our decision to pursue this path. But even a more appealing and important justification arises in the context of the learning process understanding and validation. Applications of the task-specific knowledge as the only means for inference mechanisms provide for a better understanding of the underlying principles of the learning system. Such an understanding becomes increasingly important while designing systems able not only to generate knowledge,

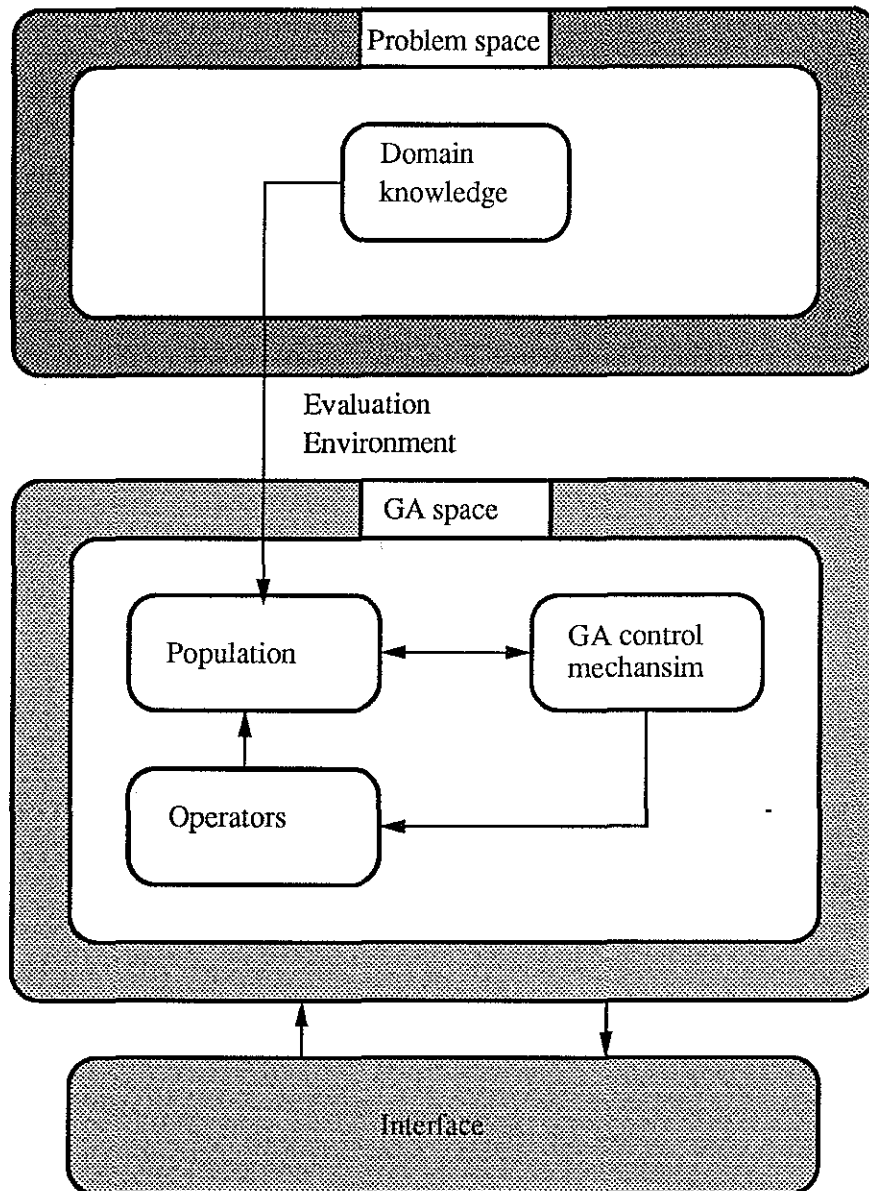


Figure 5.2: Application of a GA with traditional operators.

but also able to explain and justify their behavior. For example, Michalski wrote in [41]:

“...one should strive to facilitate human understanding not only of the surface results but also of the underlying principles, assumptions, and theories that lead to these results.”

This approach is also justified as an abstraction of the genetic algorithm approach. Following the previous discussion on GAs, and those intuitive results stating that the best representation should provide the chromosome structure reflecting syntactic and conceptual knowledge of the problem, we actually go to the extreme of using the

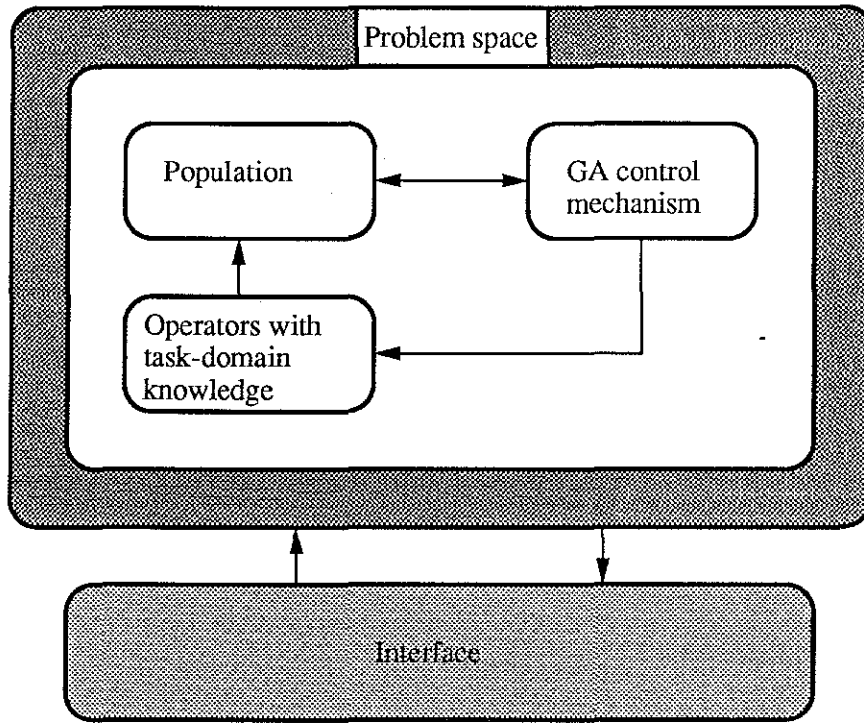


Figure 5.3: Application of a task-specific genetic algorithm.

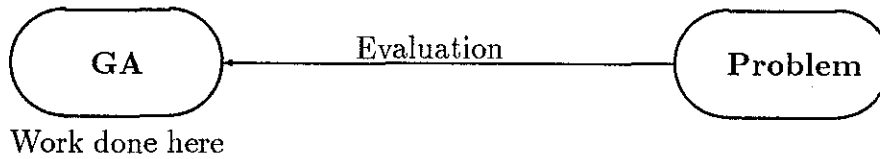


Figure 5.4: A genetic algorithm with traditional operators.

problem space as the working search space. In other words, while applications of the traditional domain-independent operators provide for a domain-independent search conducted in the artificial representation space (figures 5.4 and 5.2), we escape the critical coding problem by moving the genetic algorithm into the problem space and organizing the work there (figure 5.5 and 5.3).

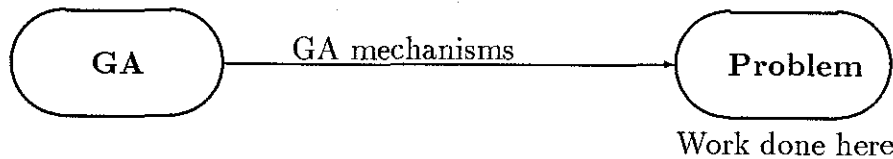


Figure 5.5: A genetic algorithm with task-specific operators.

5.2 Representation and Search Space

We adopt the multiple-valued logic language VL_1 (section 3.2) as the choice for a chromosome's representation. Then, the search space is the space of sets of rules, spanned by given features; this is the space of VL_1 concept descriptions. Because we do not employ any extra axioms, it is quite feasible and possible to have redundant descriptions, *e.g.*

$$\begin{aligned} [Age > Young] &::> HeartRiskGroup \\ [Age = Old] &::> HeartRiskGroup \end{aligned}$$

For simplicity of presentation (but not lack of generality), from now on we only consider VL_1 formulas built using the '=' relation and internal disjunction. Such assumptions do not introduce any restrictions on the power of the language (see section 3.2), and make the forthcoming discussion both more uniform and comprehensible.

Moreover, for the same reasons, we assume that we are dealing with single concepts, and that we are learning only a single description — the space not covered by such a description is assumed to represent the negation of the concept (as *e.g.* in [13]). This simplification allows us to assume a crisp rule-based conceptualization. We address possible generalizations of this approach to multiple concepts and non-crisp views in the part on future research directions (chapter 10).

Because of the assumption of learning only a single concept description, all rules are associated with the same single decision, which subsequently does not have to be stated explicitly in the "*complex ::> decision*" syntax. Accordingly, when no confusion can arise, we may refer to the same set of rules as just a logical disjunction of VL_1 complexes.

5.3 Initial Population

The population contains individuals, each of which is a potentially feasible solution (a set of rules of the VL_1 language). Its size remains fixed (as a parameter of the system). Initially the population must be filled with potential solutions. Such an initialization might be totally random (as is normally the case in genetic algorithms), or it might incorporate some task-specific knowledge. There is an obvious trade-off between the level of knowledge used in such an intelligent initialization. On one side of the spectrum is the random choice, very cheap and simple. On the other side, we have an initialization which produces actual solutions to the problem, differing possibly by some applied criteria. This latter initialization is actually as hard as the problem we wish to solve. Therefore, it is inapplicable.

We follow the idea of as simple an initialization as possible, yet intelligent. Accordingly, we allow for three different types of chromosomes to fill the population initially:

- The first type is just a random initialization. Each individual is a set of a random number of complexes, randomly generated on the search space.

- The second type is initialization with data. Each individual is just a random positive training event.
- The third type is initialization with priori hypotheses, provided such are available. Each individual is just a single hypothesis given by the environment. Having such capabilities, the system can be used as a knowledge refinement tool — possibly cooperating with an expert system of an intelligent hybrid framework.

Actual experiments show that the best average behavior is obtained while using a combination of these three (or the first two if initial hypotheses are not specified), even though the importance of the initialization with positive events diminished quite a bit when we used an operator that adds such positive events to current descriptions.

5.4 Evaluation Mechanism

The evaluation function must reflect the learning criteria. In supervised learning from examples, the criteria normally include completeness, consistency, and possibly complexity. In general, one may wish to accommodate some additional criteria, *e.g.* cost of attributes, length of descriptions, their generality, *etc.* , but we did not consider them in the current implementation.

Structure type	Completeness	Consistency
A rule set	ε^+/E^+	$1 - \varepsilon^-/E^-$
A rule	e^+/ε^+	$1 - e^-/\varepsilon^-$

Table 5.1: Completeness and consistency measures.

The completeness and consistency of a rule, or a rule set, measures its quality with respect to the set of training events. We use the formulas presented in table 5.1, where e^+/e^- is the number of positive/negative training events currently covered by a rule, $\varepsilon^+/\varepsilon^-$ is the number of such events covered by a rule set, and E^+/E^- is the total number of such events. These two measures are meaningful only to rule sets and individual rules. For conditions, the measures of the parent rule are used. These definitions assume the full memory model (see section 3.5).

Combining multiple criteria in a single evaluation measure is very difficult and critical for the convergence problem ([21]). In our case we need to combine three such values. We can ease this task by replacing the completeness and consistency measures with a single measure of *correctness*. There are two different well accepted ways to combine the two:

$$correctness = \frac{\varepsilon^+ + (E^- - \varepsilon^-)}{E^+ + E^-}$$

$$correctness = \frac{w_1 \cdot completeness + w_2 \cdot consistency}{w_1 + w_2}$$

The first of those assumes that each positive and negative event has the same effect. This may be advantageous in cases where such an assumption is true, but may be disastrous in cases where their relative number is quite different. The second, on the other hand, assumes that the positive and negative events have different effects, controlled by the weights w_1 and w_2 , and related to the relative frequency of positive/negative examples in the training set. The choice of the better measure should be based on some additional task-specific information, as what is the meaning of such relative frequencies.

We combine correctness and cost by:

$$evaluation = correctness \cdot (1 + w_3 \cdot (1 - cost))^f$$

where w_3 determines the influence of *cost* (which itself is normalized on $\langle 0, 1 \rangle$), and f grows very slowly on $\langle 0, 1 \rangle$ as the population ages (a *dynamic* approach). The cost of a description is measured by its complexity, which combines the number of rules and conditions in the following way:

$$complexity = 2 \cdot \#rules + \#conditions$$

as originally proposed by Michalski (*e.g.* [69]). The above measure for evaluation is an initial experimental rather than a theoretical choice, and provides for a controlled bias with respect to descriptions' complexity.

For most practical tests we used a very low w_3 weight (~ 0.01). Too high a value may cause lightly covered rules to be dropped from the descriptions, too low a value reduces the rate and probability of simplifying the generated descriptions (*e.g.* dropping redundant rules). The primary reason for the cost accommodation is to force differentiation between the same or similarly covering rule sets but of different complexity.

We use a dynamic approach to the use of cost *i.e.* an approach that adjusts its effects as the population ages. We successfully applied similar dynamic ideas in other domains (*e.g.* [35]). The effect of the very slowly raising f is that initially cost's influence is very light in order to promote deeper space exploration, and only increases at later stages in order to minimize descriptions' complexity. Moreover, initial experiments suggest that the system performs better when the f exponent somehow fluctuates along the desired behavior, and that the final increase should start based upon an anticipated exhaustion of resources or when the currently learned description is already complete and consistent.

5.5 Operators

The operators transform chromosomes to new (possibly better) states in the search space. Since the system operates in the problem space, the operators directly follow the inductive learning methodology. According to the three syntactic levels of the rule-based framework (conditions, rules, rule sets), we divide the operators into three corresponding groups. In addition, each operator is classified as having either generalizing (subsequently denoted by \triangleleft), specializing (denoted by \triangleright), or unspecified —

or independent — behavior (denoted by \diamond). Note that the inductive methodology (section 3.6) does not define independent operators; their introduction is strongly associated with the ideas of this genetical search, *e.g.* the use of a population.

For a graphical illustration of the operators, we use the following search space:

Attribute	Values	Type
<i>A</i>	0,1,2,3	Linear
<i>B</i>	0,1	Nominal
<i>C</i>	0,1,2	Nominal
<i>D</i>	0,1	Nominal

in which we demonstrate each operator using the idea of *diagramatic visualization* ([69]), which is a multi-valued extension of the well known “Karnaugh map” or “Veitch diagram”. Following the correspondence of VL_1 complexes and rules in the single-concept scenario, we represent a rule set as a disjunction of VL_1 complexes; each complex is a *lhs* of a rule corresponding to the same decision. We discuss possible generalizations to multiple decisions in section 10.2. Also, we try to define the operators as simple as possible, in order to reduce the computational overhead. For example, we do not use the inductive resolution rule (section 3.6), which requires an extensive pattern matching in order to find two complexes having selectors which are negations of each other.

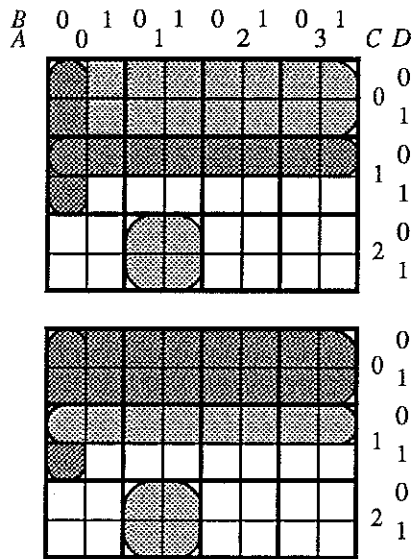
In the subsequent definitions we use the following notation: *chrom* for a chromosome representing a set of rules, *cpx* for a complex, *sel* for a selector, *dec* for a decision, and e^+/e^- for a positive/negative example from a category.

5.5.1 Definitions

Rule set (VL_1 set of complexes) level. At this level, operators act on whole rule sets (one or two at a time):

- Independent:

Rules exchange. This operator requires two parent rule sets, and it exchanges random rules between these two. It requires two parameters: probability of application to a rule set, and probability of rules selection for the exchange.



Parent rule set 1:
 $[A=0][B=0][C=0,1] \vee [C=1][D=0]$

Parent rule set 2:
 $[C=0] \vee [A=1][C=2]$

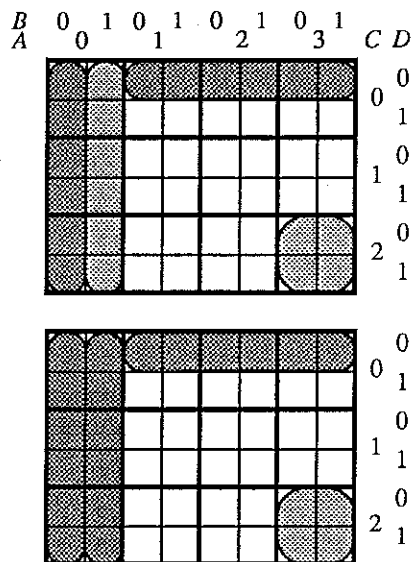
Consider the case of exchanging the second rule from parent 1 with the first rule of parent 2

Offspring 1:
 $[A=0][B=0][C=0,1] \vee [C=0]$

Offspring 2:
 $[A=1][C=2] \vee [C=1][D=0]$

- Generalization:

Rules copy. This operator requires two parent rule sets, and it copies a random rule from each of the the sets to the other. It differs from the “rules exchange” operator, as it does not remove information being propagated from the rule set. It requires one parameter: probability of application to a rule set.



Parent rule set 1:
 $[A=0][B=0] \vee [A=1,2,3][C=0][D=0]$

Parent rule set 2:
 $[A=0][B=1] \vee [A=3][C=2]$

Consider the case of copying the second rule of parent 2 to parent 1

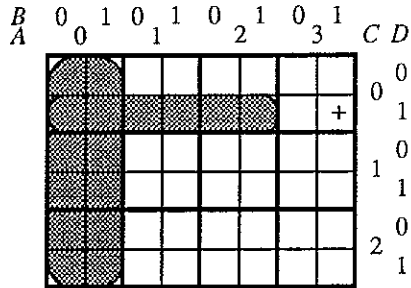
Offspring 1:
 $[A=0][B=0] \vee [A=0][B=1]$
 $\vee [A=1,2,3][C=0][D=0]$

Offspring 2:
 $[A=0][B=1] \vee [A=3][C=2]$

New event. This operator acts on a single rule set: if there is a positive event not covered yet by the current rule set, this event's description is added to the set as a new rule:

$$chrom = \bigcup_i (cpx_i ::> dec) \text{ and } \exists_{e^+} \forall_i (e^+ \not\Rightarrow cpx_i) \triangleleft chrom \cup (e^+ ::> dec)$$

It requires only one parameter: probability of application to a rule set.

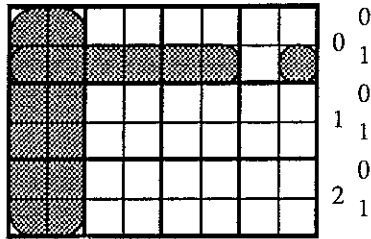


Parent rule set:

$$[A=0] \vee [C=0][D=1]$$

An uncovered event:

$$[A=3][B=1][C=0][D=1]$$



Offspring:

$$[A=0] \vee [C=0][D=1]$$

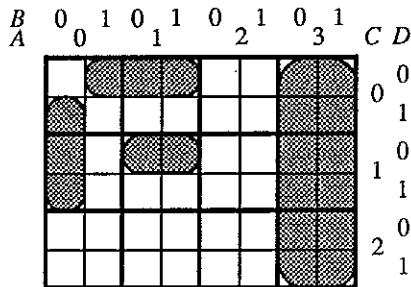
$$\vee [A=3][B=1][C=0][D=1]$$

Rules generalization. This operator acts on a single rule set. It selects two random rules and replaces them by their most specific generalization:

$$cpx_1 ::> dec, cpx_2 ::> dec \triangleleft (cpx' ::> dec)$$

where cpx' is the most specific generalization (not necessarily consistent with respect to previously excluded negative events) of the two complexes.

It requires one parameter: probability of application to a rule set.



Parent rule set:

$$[A=3] \vee [A=1][C=1][D=0]$$

$$\vee [A=0][B=0][C=1]$$

$$\vee [A=1][C=0][D=0]$$

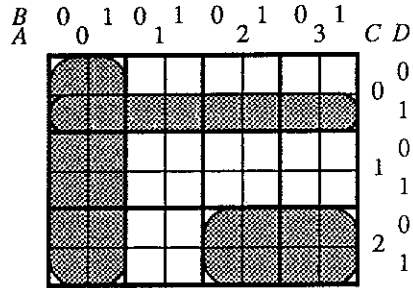
Consider generalizing all but the the first rule

Offspring:

$$[A=3] \vee [A=0,1][C=0,1]$$

- Specialization:

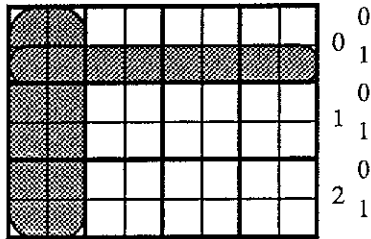
Rules drop. This operator acts on a single rule set, and it drops a random rule from that set. It requires one parameter: probability of application to a rule.



Parent rule set:

$$[A=0] \vee [C=0][D=1] \vee [A=2,3][C=2]$$

Consider dropping the third rule



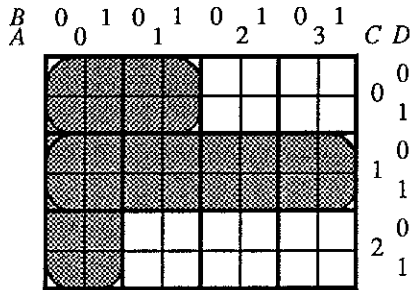
Offspring:

$$[A=0] \vee [C=0][D=1]$$

Rules specialization. This operator acts on a single rule set, and it replaces two random rules by their most general specification:

$$cpx_1 ::> dec, cpx_2 ::> dec \triangleright (cpx' ::> dec)$$

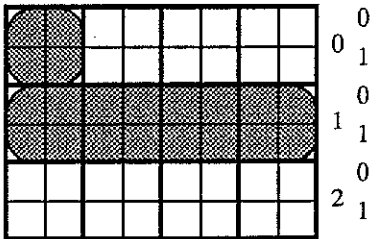
where cpx' is the most specific generalization (not necessarily complete) of the two complexes. It requires one parameter: probability of application to a rule set.



Parent rule set:

$$[A=0] \vee [A=0,1][C=0] \vee [C=1]$$

Consider specializing the first two rules



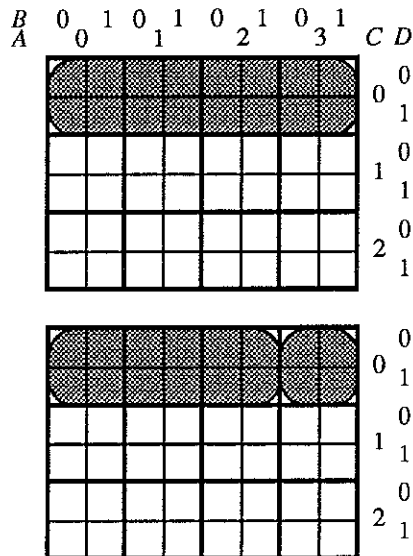
Offspring:

$$[A=0][C=0] \vee [C=1]$$

Rule (VL_1 complex) level. At this level, operators act on one rule at a time:

- Independent:

Rule split. This operator acts on a single rule, and it splits it into a number of rules, according to a partition of the values of a condition (an absent condition can be selected as well, using all domain values). The set of domain values present in the selected condition can be split according to each value individually or according to two disjoint subsets of values. For the linear data types, the latter split is more desired; in this case the present domain values are split by cutting the ordered set of values in a single random place. For the nominal data type, the former split is more desired. A structured type requires a slightly more sophisticated approach, similar to that of the linear type, but with differently defined values and orderings. This operator requires three parameters: probability of application to a rule, and probabilities of a subset vs. all values split, separately for the linear and nominal data types.



Parent rule:

$$[C=0]$$

Assume the attribute A is linear, and a subset split occurs between the values 2 and 3

Offspring:

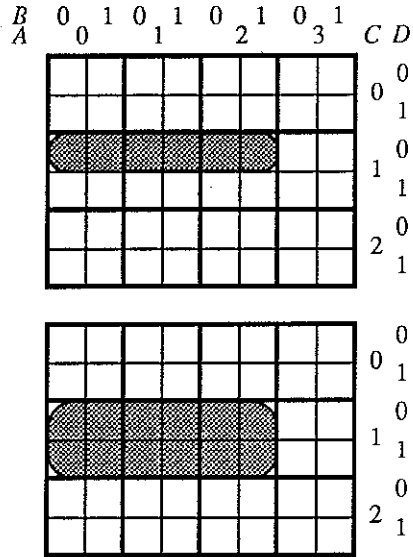
$$[A=0,1,2][C=0] \vee [A=3][C=0]$$

- Generalization:

Condition drop. This operator acts on a single rule, and it removes a present condition from that rule:

$$((cpx = \bigwedge_i sel_i) ::> dec) \triangleleft (cpx' ::> dec)$$

where cpx' has all but one selectors of cpx . In other words, one of the selectors of cpx is extended to cover the whole domain of the associated attribute. It requires a single parameter: probability of application to a rule.



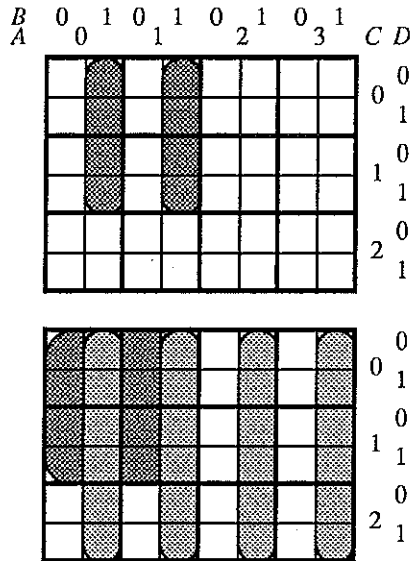
Parent rule:
 $[A=0,1,2][C=1][D=0]$

Assume that the condition associated with the D attribute is dropped

Offspring:
 $[A=0,1,2][C=1]$

Turning conjunction into disjunction. This operator acts on a single rule, and it splits the complex into a disjunction:

$((\bigwedge_i sel_i \wedge \bigwedge_j sel_j) ::> dec) \triangleleft (\bigwedge_i sel_i ::> dec) \vee (\bigwedge_j sel_j ::> dec)$
 where the complex's separation into n and m selectors is random and position independent. It requires a single parameter: probability of application to a rule.



Parent rule:
 $[A=0,1][B=1][C=0,1]$

Offspring:
 $[A=0,1][C=0,1] \vee [B=1]$

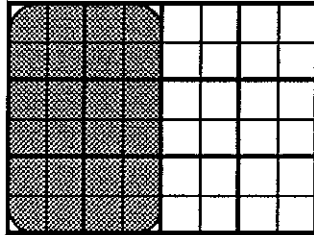
- Specialization:

Condition introduce. This operator acts on a single rule, and it introduces a random condition associated with an unconditioned attribute:

$((cpx = \bigwedge_i sel_i) ::> dec) \triangleright (cpx' ::> dec)$
 where cpx' has, in addition to all the selectors of cpx , a new selector associated with an attribute not present in cpx . The new selector is a random

choice from among all of its possible internal disjunctions. It requires a single parameter: probability of application to a rule.

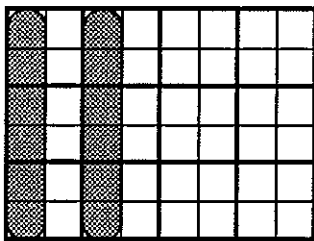
B 0 1 0 1 0 1 0 1
A 0 1 2 3 C D



Parent rule:

$$[A=0,1]$$

Assume that we introduce a new condition on attribute B: $[B=0]$



Offspring:

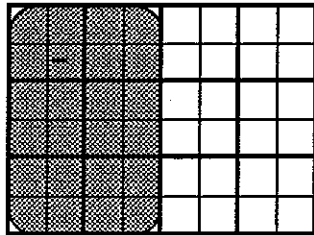
$$[A=0,1][B=0]$$

Rule directed split. This operator acts on a single rule. If this rule covers a negative event, it is split into a set of maximally general rules, yet consistent with that event, in the following way:

$$(cpx ::> dec) \text{ and } \exists_{e^-} (e^- \Rightarrow cpx) \triangleright \bigcup_i (cpx_i ::> dec)$$

where the new set has cpx 's such that $(\bigvee_i cpx_i) = (cpx \wedge \neg e^-)$. This operator resembles the action in the heart of the cover procedure of AQ15. It requires a single parameter: probability of application to a rule.

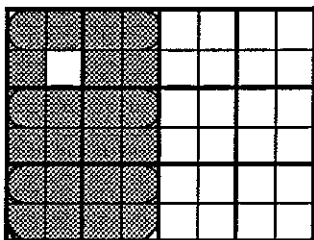
B 0 1 0 1 0 1 0 1
A 0 1 2 3 C D



Parent rule:

$$[A=0,1]$$

$$e^- = [A=0][B=1][C=0][D=1]$$



Offspring:

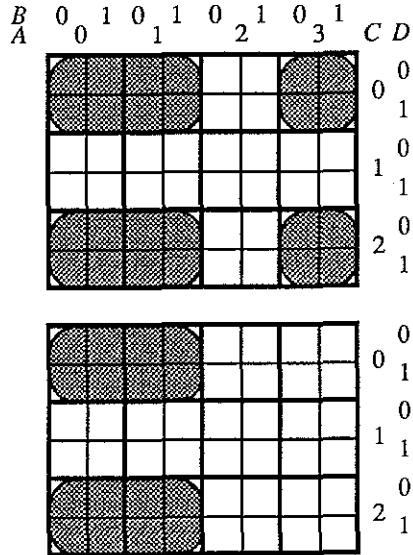
$$[A=1] \vee [A=0,1][B=0]$$

$$\vee [A=0,1][C=1,2] \vee [A=0,1][D=0]$$

Condition (VL_1 selector) level:

- Independent:

Reference change. This operator acts on a single condition, and it randomly removes or adds a single domain value to this condition. It requires a single parameter: probability of application to a condition.



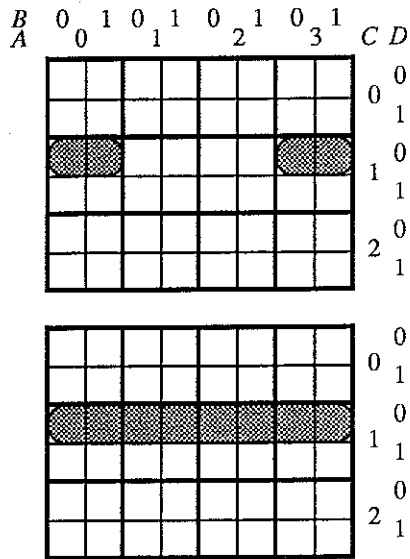
Parent rule:
 $[A=0,1,3][C=0,2]$

Assume the value '3' is removed from the condition associated with the attribute A

Offspring:
 $[A=0,1][C=0,2]$

- Generalization:

Reference extension. This operator acts on a single condition, and it extends the domain by allowing a number of additional values. For the nominal type, some random values are selected for extension. For the linear type, a single value may be selected or, with a higher chance, a range may be closed (between two present values, using the domain ordering). Moreover, such shorter open ranges have a higher chance of being selected over longer ones. This operator requires quite a few parameters, including a probability of application to a condition, and a number of selection probabilities determining the choice of an action. For the structured type, we replace some of the present values by their parent in the generalization tree, giving preference to those offsprings that prevail in number (generalization climbing).



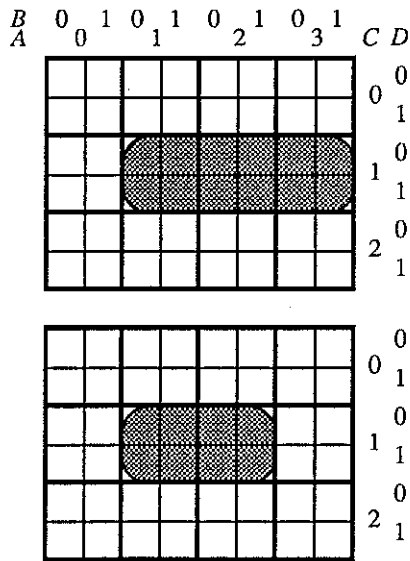
Parent rule:
 $[A=0,3][C=1][D=0]$

Assume we work with the condition on the linear attribute A , and we close the open interval

Offspring:
 $[C=1][D=0]$

• Specialization:

Reference restriction. This operator acts on a single condition, and it removes some domain values from this condition. Its actions and parameters are analogous to those of the “reference extension”, but have opposite effects.



Parent rule:
 $[A=1,2,3][C=1]$

Assume we work with the condition on the attribute A , and we remove the value '3' from the condition on the attribute A

Offspring:
 $[A=1,2][C=1]$

5.5.2 Dynamic aspects

As seen in the definitions, each operator is given some initial probabilities from two separate groups:

- *Application probabilities.* Each operator is given an initial probability of application to its type of structure (based on the level of its definition). These

probabilities have a dynamic character with respect to the current context, *i.e.* to both the current coverage and the current, problem-dependent, size of the average chromosome. Firstly, priori probabilities of generalizing operators are increased for applications to structures (rule sets, rules, conditions) that are incomplete, and decreased for those inconsistent. On the other hand, the priori probabilities of specializing operators are increased for applications to structures that are inconsistent, and decreased for those that are incomplete. Moreover, the levels of probability increase/decrease are based on the levels of inconsistency/incompleteness. In other words, these two measures serve as additional heuristics guiding the selection of appropriate operators (in addition to fitness). Secondly, all application probabilities are adjusted as to achieve a constant chromosomes' update rate. For example, more complex problems, which cause the intermediate chromosomes to be longer (both in terms of the number of complexes and their sizes), decrease all such probabilities by the same fraction.

- *Selection probabilities:* These serve as a mean of selecting one of a number of possible actions or substructures to participate in the operations — they are static.

While selecting the appropriate method of completeness/consistency incorporation, we must be careful not to decrease the probabilities too far as to prevent certain operations from performing. Since we want the changes to be linear with those measures, the following seem natural (but still experimental) choices:

Generalizing operators: $p' = p \cdot (\frac{3}{2} - \text{completeness}) \cdot (\frac{1}{2} + \text{consistency})$

Specializing operators: $p' = p \cdot (\frac{1}{2} + \text{completeness}) \cdot (\frac{3}{2} - \text{consistency})$

The new value p' is the adjusted probability, and p is the actual probability. It is important to mention that since p' is computed differently for each chromosome, it does not replace the a priori p . The simplicity of this formulas guarantee low computational overhead.

To accommodate the changes in problem-specific characteristics, namely the average size of a complex and the average length of chromosomes in the current population, we use the following approximation. We observe the number of chromosomes undergoing recombination in a given population. If this number represents too large a portion of the population, we decrease all the priori application probabilities by the same fraction for the next reproductive iteration. If this number is too small, we do the opposite. Since the adjustments are computed for the whole population, they actually always replace the priori values (or those resulting from the last change). Experiments show that with an appropriately small such adjustment fraction, the changes converge and then the probabilities remain relatively steady. This method provides for a partial independence of such probabilities from some characteristics of the problems. We discuss more such methods in section 10.1.

5.6 Algorithm

The algorithm uses the above components and the control of genetic algorithms (section 2.1). At each iteration all rule sets of the population are evaluated, and a new population is formed by drawing members from the original one in such a way that more fit individuals have a higher chance of being selected (chapter 2 and 5.4). Following that, the operators are applied to such a population in order to move these partial solutions, hopefully, closer to the sought state. Each operator acts on structures from its level, applying itself to some randomly selected structures: the application depends on both initial probabilities, the consistency/completeness of such structures, and the size of the currently average chromosome. Then, the cycle repeats until a desired description is found or some resources are exhausted. For a better illustration refer to chapter 7.

Chapter 6

SOME IMPLEMENTATION ISSUES

We implemented a simple C version of the proposed approach in order to be able to test our ideas. The choice of the language, aside from efficiency, was based on availability of bitwise logical operators used to speed up the evaluation mechanism (section 6.3). We call this implementation **GIL** (for **Genetic-Based Inductive Learning**). In this chapter we present the most important issues facing any such implementation, along with approaches used in GIL.

6.1 Sampling Mechanism

The selection algorithm is to choose some chromosomes from the current population to form a new population, with possible omissions and repetitions. There are many standard ways of performing this step. Baker provides an excellent discussion of such in [3]: we use the *stochastic universal sampling* mechanism. This method builds a roulette wheel for the chromosomes, with each chromosome having allocated a portion of the wheel proportional to its evaluation fitness. A second wheel is constructed with equally spaced marks. The number of such marks is the same as the number of samples to be drawn (the size of the population). The wheels are placed on a single axis and the one with marks is randomly spun against the other. The positions of the marks, in relation to the space allocated for each chromosome on the other wheel, are then observed: a chromosome is selected once for each mark landing in its allocated space.

6.2 Internal Representation

In section 5.2 we described the architecture of the chromosome, in terms of the language used. Now we discuss some important issues associated with the internal representation.

In section 4.2 we mentioned that the widely used three-symbol alphabet is not suitable for the multi-valued domains of feature-based spaces. A more appropriate solution was suggested originally by Greene and Smith in [17], and recently used by

DeJong and Spears in their GABIL system ([13]). This approach uses binary digits to represent domain values. For example, assuming that an attribute has five domain values, the binary vector 11001 represents the condition saying that the attribute must have the first, second, or the last value (assuming some positional enumeration of values from the left, and a use of the internal disjunction). If the longest domain is not longer than the bitwise length of available integer or unsigned data types, each condition can easily be represented by one such simple data object. If some domains are longer, a vector of such data objects can be used. We use exactly these ideas to implement conditions (selectors of VL_1).

In section 3.2 and 5.2 we showed how, without the loss of generality, our assumption of learning only one description allows us to work with complexes built using the internal disjunction and only the '=' relation, and that a disjunction of such complexes can be treated interchangeably with the a set of VL_1 rules. We use this fact throughout this implementation, and we discuss possible extensions to multiple descriptions in section 10.2.

A chromosome's length is actually unrestricted — a rule set may contain any number of rules. Because of that, the only way to represent rule sets is to organize them as linked lists. This gives us freedom up to a given machine's capacity. An important issue associated with the rule set is that of treating rules that are *invalid*, *i.e.* conditions that are totally restricted — exclude all domain values. DeJong and Spears suggested keeping such rules as possible sources of valid conditions (GABIL uses this strategy). We performed some experiments to study the trade-off between anticipated increases in the computational cost of retaining these rules vs. improvements in predictive accuracy of the system. Our conclusions are far from final. Nevertheless, they suggest there is a clear conflict between these two factors (see section 9.6). A similar issue arises in the context of *empty* rules, *i.e.* those not covering any positive events. Such rules, again, may be removed or retained. Also, there seems to be a similar kind of complexity vs. quality trade-off. We experimented with this issue as well, and some results are reported in the same section.

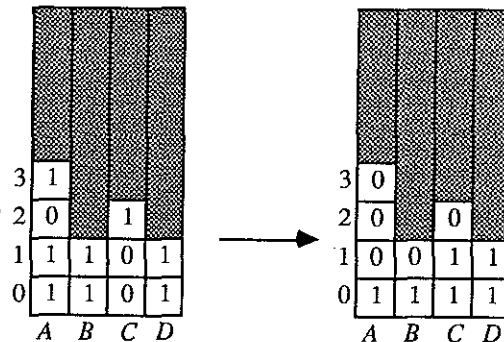


Figure 6.1: An internal representation of a sample chromosome.

Each complex is a conjunction of a number of conditions. The number of such possible conditions, in a given complex, is bounded by the total number of attributes. We use that bound as a way of simplifying the internal representation: a complex is

represented by a vector of conditions. Furthermore, for simplicity and efficiency, we associate a fixed positional correspondence between the attributes of the vector. This does not introduce any problems since no operators acting at the condition level are positionally dependent — their application is nondeterministic.

Such an implementation introduces a new dilemma: how to treat unrestricted conditions, *i.e.* those that include all domain values in the selector. It is a question of elegance, or possible efficiency, rather than power: an unrestricted selector can be dropped from its complex without of any semantic change to the rule associated with this complex. We tried both approaches: one with all selectors present in each complex, the other with unrestricted selectors invisible to all operators by means of a special flag. Since we found no significant difference in the system's performance under both conditions, from now on we assume the latter approach without a loss of generality.

The above ideas are illustrated in figure 6.1, assuming the set of features used in section 5.5 while defining the operators and an 8-bit machine. This figure shows an internal representation of a chromosome, which can be viewed as a disjunction of complexes or a set of rules using an implicit decision. Moreover, depending on the treatment of unrestricted domains, the same chromosome can be described in different ways, as well. The resulting four possible semantically equivalent views are as follows:

- $[A = 0, 1, 3][B = 0, 1][C = 2][D = 0, 1] \text{ ::> } dec,$
 $[A = 0][B = 0][C = 0, 1][D = 0, 1] \text{ ::> } dec$
- $[A = 0, 1, 3][C = 2] \text{ ::> } dec,$
 $[A = 0][B = 0][C = 0, 1] \text{ ::> } dec$
- $[A = 0, 1, 3][B = 0, 1][C = 2][D = 0, 1] \vee [A = 0][B = 0][C = 0, 1][D = 0, 1]$
- $[A = 0, 1, 3][C = 2] \vee [A = 0][B = 0][C = 0, 1]$

6.3 Data Compilation

A very commonly cited disadvantage of genetic approaches to problem solving is their time complexity (*e.g.* [52]). This problem becomes especially visible when the evaluation requires an extensive computation. This is also the case when evaluating rule sets in the supervised inductive learning, as this process involves extensive pattern matching. Concerned with such problems, we designed a special method of data compilation, aimed at improving the time complexity of the system.

The idea is as follows: rather than storing data in terms of features, store features in terms of data coverage (assuming full memory learning). In other words, for each possible feature, retain information about the events covered by this feature. This must be done separately for each concept. Moreover, it must be done even for concepts not being explicitly learned. For example, this means that GIL has to remember such coverage separately for both the concept and its negation. We achieve this by enumerating all learning events, and constructing binary coverage vectors.

Positive coverage vector: 10000001000000001010000000
 Negative coverage vector: 000000000001010

Figure 6.2: Examples of binary coverage vectors of a feature.

The idea behind these vectors is analogous to that of representing conditions. Suppose a learning session uses E^+ positive events and E^- negative examples. Then, a coverage vector is constructed as a vector of a simple data type (integer or unsigned) of length $l = \lceil E / (8 \cdot \text{sizeof}(\text{datatype})) \rceil$, for both E^+ and E^- separately. In such a vector, a binary one at position n indicates that the structure that owns this coverage vector covers event $\#n$. For example, the vectors in figure 6.2 indicate that the given feature covers positive events $\#1, 8, 17, 19$ (out of 25), and negative events $\#12$ and 14 (out of 15).

As mentioned, prior to learning all data is precompiled into such vectors, for all possible features. During the actual run of the system, similar vectors are constructed for all structures of the database: from the features upwards. For example, having the feature coverages we can easily construct both positive and negative coverage of the condition $[A=0,2]$ by means a simple bitwise OR (assuming a language that provides such an operation, *e.g.* C) on appropriate coverage vectors of features ($A=0$) and ($A=2$). Subsequently, conditions' coverages are propagated to rules by means of a simple bitwise AND. Finally, rules' coverages are propagated to rule sets again by means of the bitwise OR.

Perhaps the most important effect of such an approach is that we can easily incrementally upgrade such coverages using a minimal amount of work after the initial database is fully covered. For example, consider a case of the rules **copy** operator applied to the following two rule sets:

$$R_1 = r_1^1, r_1^2$$

$$R_2 = r_2^1, r_2^2, r_2^3$$

and suppose the operator copies r_2^2 to R_1 . The coverage of the second rule set does not change. To compute the coverage of the first rule set it is sufficient to perform bitwise OR between the coverage of the rule r_2^2 (which did not change during this operation) with the coverage of the original R_1 . In other words, we compute this coverage using two bitwise OR operations (one for the positive and one for the negative coverage). In general, the number of such required operations increases (very slowly linearly) with the number of training events.

As another example, consider the case of the **reference change** operation, with a single change from 0 to 1, on position $\#$, in a condition's binary vector. All that needs to be done to update the coverage of this condition is to perform bitwise OR on the coverages (positive and negative) of the corresponding feature number $\#$ associated with the given attribute with those of the original condition. Then, this change must be propagated to the appropriate rule's and rule set's coverages, using similar simple computations.

Chapter 7

A TRACE OF THE SYSTEM'S BEHAVIOR

In this chapter we trace GIL's behavior on a sample application. We explain in detail the major steps of the GA algorithm (figure 2.1): initialization, initial evaluation, and one basic iteration. Finally, we trace all the remaining iterations. This further explains and exemplifies this system and its ideas. For this experiment, the following implementation options were used: delete both invalid and empty complexes, make the unrestricted selectors invisible, and use the correctness measure that combines completeness and consistency with the equal weights.

For the experiment, we decided to use data of a moderate complexity, yet simple enough to be represented by the diagrammatic visualization method. One of the concepts further described and tested in section 9.2 was a perfect choice. It comes from the world of Emerald's robots ([29]) described by the following six attributes (we boldface the abbreviations subsequently used in this chapter):

Attribute	Values
<i>HeadShape</i>	Round, Square, Octagon
<i>Body</i>	Round, Square, Octagon
<i>Smiling</i>	Yes, No
<i>Holding</i>	Sword, Balloon, Flag
<i>JacketColor</i>	Red, Yellow, Green, Blue
<i>Tie</i>	Yes, No

and is one of those presented in [69]:

Head is Round and Color is Red or Head is Square and Holding a Balloon

The above concept is represented by the following set of rules:

$[H=R][J=R] \Rightarrow \textit{Concept}$
 $[Ho=S][H=B] \Rightarrow \textit{Concept}$

or, assuming an implicit decision as used in GIL, by the following formula:

$[H=R][J=R] \vee [H=S][Ho=B]$

The goal was to learn such a concept description when only presented with a limited number of training events. In other words, this experiment was designed to test both GIL's ability to learn descriptions and its ability to generalize.

The above attributes span an event space of size 432: 84 of them satisfy the concept. The training was done using a random 20% of both positive and negative examples: 17 and 70 respectively (see figure 7.1 for a visualization of the target concept and the training events). The population size was set to 40, initialized equally by both random descriptions and positive training events. The system was set to run 100 iterations. Other implementational parameters were set as follows: $w_1 = w_2 = 0.5$, $w_3 = 0.02$, the *cost* was normalized with respect to the highest cost in the current population. The priori application probabilities, along with actual adjusted values (adjustment for the currently average size of the chromosomes, see section 5.4) at the end of this experiment, and at the end of one of those of chapter 9, are presented in table 7.1. This scaling was computed assuming a desired rate of 80% chromosomes to be updated by the recombination step. The selection probabilities were as follows: 0.2 for a rule selection in "rules exchange"; 0.1 for "splitting a rule" according to two subsets, as opposed to all domain values, for the nominal type, and 0.7 for the linear type; 0.5 for all probabilities on the condition level.

Level	Operator	Initial Values	This experiment	Multiplexer f_{11}
Rule set	Rules exchange	0.20	0.115	0.011
	Rules copy	0.10	0.058	0.050
	New event	0.40	0.230	0.022
	Rules generalization	0.50	0.288	0.027
	Rules drop	0.50	0.288	0.027
	Rules specialization	0.50	0.288	0.027
Rule	Rule split	0.02	0.011	0.002
	Condition drop	0.10	0.058	0.005
	Turning conj. into disjunc.	0.02	0.011	0.002
	Condition introduce	0.10	0.058	0.005
	Rule directed split	0.12	0.069	0.007
Condition	Reference change	0.02	0.012	0.001
	Reference extension	0.03	0.017	0.002
	Reference restriction	0.03	0.017	0.002

Table 7.1: Application probabilities.

7.1 Data Compilation

Data compilation is not part of the algorithm itself, but was rather designed and introduced as an implementation method for improving the efficiency of the evaluation

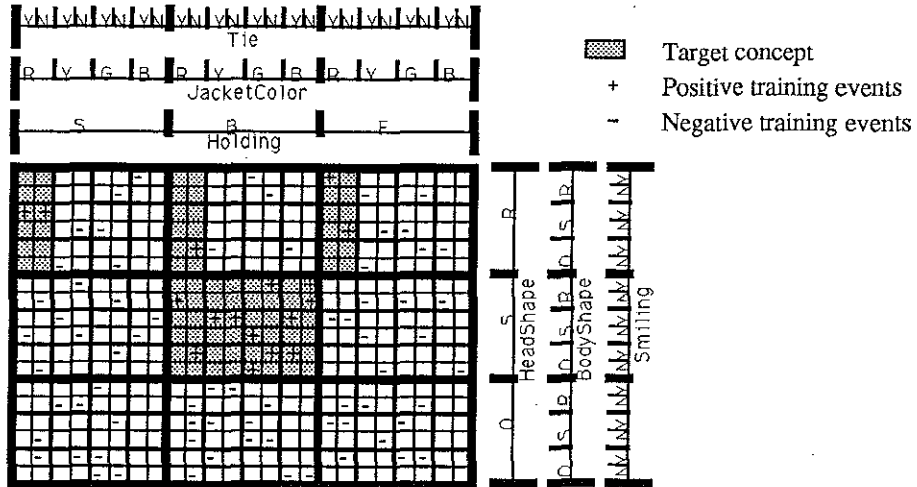


Figure 7.1: The goal concept and the training events.

mechanism. This method is described in section 6.3: all the training events were compiled into positive and negative binary coverage vectors for all possible features of the event space.

7.2 Initialization

The population size was set at 40, *i.e.* the system was simultaneously working on that many different potential solutions. Initially, half of these chromosomes were set as random positive events (with replacement). The other half was initialized to random disjunctions of complexes in the following way:

- Make one random complex.
- Append another random complex with a certain probability (0.7 was used).
- Repeat the previous step until no complex is appended.

The probability of appending a new complex controls the expected value of the length of such random chromosomes.

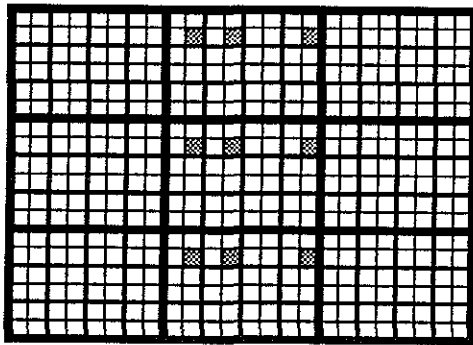
7.3 Initial Evaluation

Each chromosome, being a disjunction of complexes, was initially evaluated in the following way:

- Both positive and negative binary coverage vectors of each selector present in the population were constructed by bitwise ORing the coverage vectors of appropriate features. For example, to construct the positive coverage vector for the selector $[H=R, O]$, the system would OR such positive vectors of the features $H=R$ and $H=O$.

- Both positive and negative binary coverage vectors of each complex were analogously constructed by ANDing appropriate vectors of its selectors.
- The number of binary ones in each such a vector determined the current e^+ and e^- coverage of the corresponding complex.
- Both positive and negative binary coverage vectors of each chromosome were analogously constructed by ORing appropriate vectors of its complexes.
- The number of binary ones in each such a vector determined the current ϵ^+ and ϵ^- coverage of the corresponding chromosome.
- Both completeness and consistency measures were calculated for all chromosomes and complexes, according to section 5.4.
- The total fitness of each chromosome was evaluated according to section 5.4.

The best such evaluated chromosome of the initial population is presented in figure 7.2; it is a result of a random chromosome generation.



Iteration 0 (from initialization)
 Cost = 7 : 1 rule, 5 conditions
 Positive coverage = 1
 Negative coverage = 0
 Best rules:
 $[B=R][S=N][Ho=B][J=R, Y, B][T=N]$

Figure 7.2: The best initial chromosome.

7.4 One Iteration

Each iteration of the genetic algorithm consists of three basic steps: selection, reproduction, and evaluation (see figure 2.1). However, the special data compilation method, along with the use of the binary coverage vectors, allows for combining the last two steps into an incrementally evaluated reproduction: each operator is followed by a proper update to all the affected vectors and recalculation of the completeness and consistency measures (see section 6.3). Then, the only task of the evaluation step is to calculate the total fitness: there is no pattern matching involved. Nevertheless, such a method still follows the full memory approach: the actual data is replaced by the coverage vectors of the features. In the case of an incremental learning (this particular example was conducted in the batch mode), every time a new example is presented to the system, all appropriate features (those present in that example)

would be incrementally updated and propagated to other structures present in the population.

7.4.1 Selection

During this step a new population is selected from members of the previous population with replacement (or from the initial population during the first iteration). The selection is performed using the stochastic universal sampling mechanism described in section 6.1, which increases selection chances for higher-evaluated chromosomes. One important implication of this mechanism is that the best chromosome is always guaranteed to appear at least once in the next population.

7.4.2 Reproduction

During this step some of the chromosomes of the new population are changed by actions of the genetic operators. Normally, such actions are selected based on some static probabilities. However, in our case there are two dynamic factors that affect such probabilities: the rate of chromosomes update and completeness and consistency measures of proper structures.

First, the rate of chromosomes update (from the previous generation) is compared to that desired (80% in this case, or 32 chromosomes), and if they differ by more than some allowable margin, all the application probabilities are accordingly adjusted by a small fraction and the new values replace the old ones. Then, the algorithm's control walks through all the structures present in the population and nondeterministically applies selected operators. Competing here operators are those defined for the same level. Moreover, both the generalizing and the specializing operators adjust their application probabilities for each structure, according to section 5.5.2, before a uniform probability generator decides their actual application. For example, consider a complex with the following measures: *completeness* = 0.2, *consistency* = 0.9. All operators defined for this level are tried in a random order: each one actually found applicable updates this complex. The independent operators have probabilities of application exactly as the priori values (possibly adjusted with respect to the desired update rate). The generalizing operators have probabilities of application additionally adjusted by

$$\left(\frac{3}{2} - \text{completeness}\right) \cdot \left(\frac{1}{2} + \text{consistency}\right) = 1.82$$

and the specializing operators have the probabilities adjusted by

$$\left(\frac{1}{2} + \text{completeness}\right) \cdot \left(\frac{3}{2} - \text{consistency}\right) = 0.42$$

In other words, this particular complex would have an increasing pressure for generalization.

Each operator actually selected for application updates the structure, and then it immediately incrementally updates binary coverage vectors from the structure up to the chromosome level. This action is very cheap at this moment for most of the

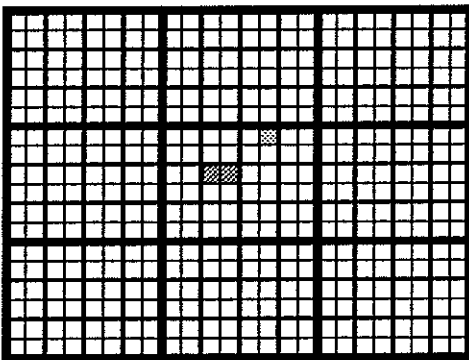
operators (for some examples see section 6.3). Also, the appropriate completeness and consistency measures are immediately reevaluated. Such an incremental approach not only reduces the time complexity of evaluations by taking into account some specific information about properties of the operators, but also leaves all the coverages and the measures consistent for the other competing operators. This is very important since some of them rely on such information for further efficiency improvements. For example, the “directed rule split” operator needs to find a negative event inconsistent with the current complex. Being able to rely on the coverage vectors, finding such an event is reduced to selecting a random binary one from the negative coverage vector of this complex: otherwise, the complex would have to be sequentially matched against all possible negative training examples.

7.4.3 Evaluation

As we mentioned above, the only task left for the evaluation is to update the fitness values, using the completeness, consistency, and cost measures of each chromosome, as well as the maximal cost found in this current population (see section 5.4).

7.5 The Remaining Iterations

Following the initialization and the initial evaluation (sections 7.2 and 7.3), the system was run for the allowable number of iterations. Not every iteration produced a better chromosome. In this section we show the best chromosome at each iteration that generated such an improvement.



Iteration 1

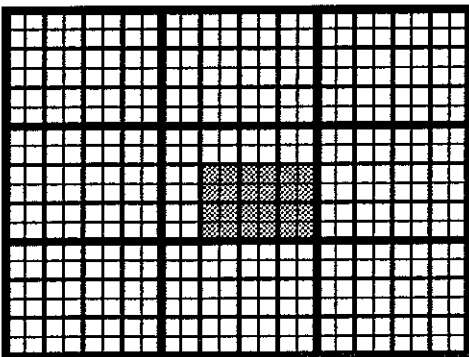
Cost = 15 : 2 rules, 11 conditions

Positive coverage = 3

Negative coverage = 0

Best rules:

$$[H=S][B=S][S=Y][Ho=B][J=Y]$$

$$\vee [H=S][B=R][S=Y][Ho=B][J=G][T=N]$$


Iteration 2

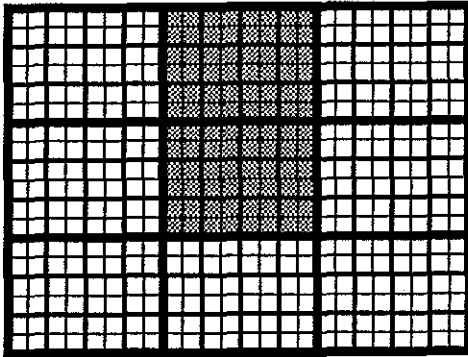
Cost = 6 : 1 rule, 4 conditions

Positive coverage = 6

Negative coverage = 0

Best rules:

$$[H=S][B=S,O][S=Y][Ho=B][J=Y,G,B]$$



Iteration 3 (an overgeneralization)

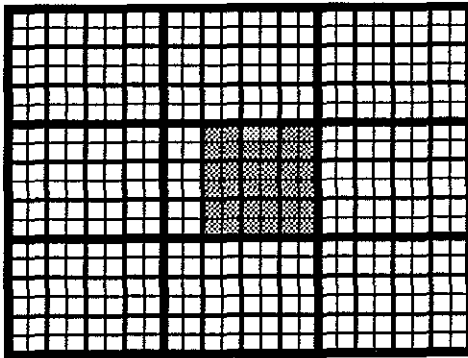
Cost = 4 : 1 rule, 2 conditions

Positive coverage = 12

Negative coverage = 4

Best rules:

$[H=R,S][Ho=B]$



Iteration 7 (a redundant rule)

Cost = 12 : 2 rules, 8 conditions

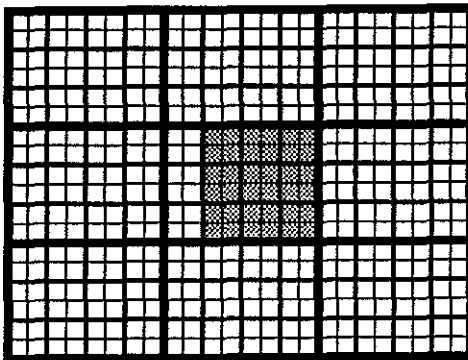
Positive coverage = 10

Negative coverage = 0

Best rules:

$[H=S][Ho=B][J=Y,G,B]$

$\vee [H=S][B=R][S=Y][Ho=B][J=G]$



Iteration 8 (the redundancy removed)

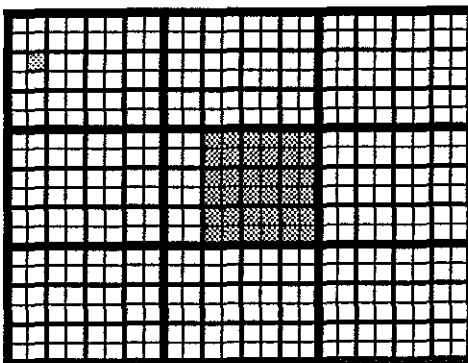
Cost = 5 : 1 rule, 3 conditions

Positive coverage = 10

Negative coverage = 0

Best rules:

$[H=S][Ho=B][J=Y,G,B]$



Iteration 9

Cost = 13 : 2 rules, 9 conditions

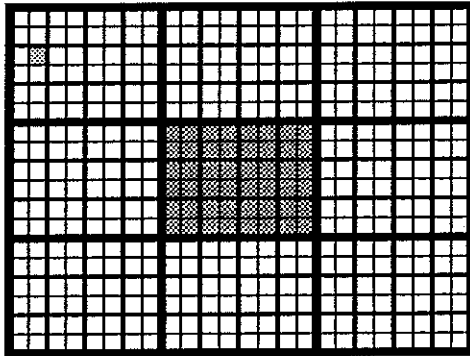
Positive coverage = 11

Negative coverage = 0

Best rules:

$[H=S][Ho=B][J=Y,G,B]$

$\vee [H=R][B=S][S=Y][Ho=S][J=R][T=N]$



Iteration 10

Cost = 12 : 2 rules, 8 conditions

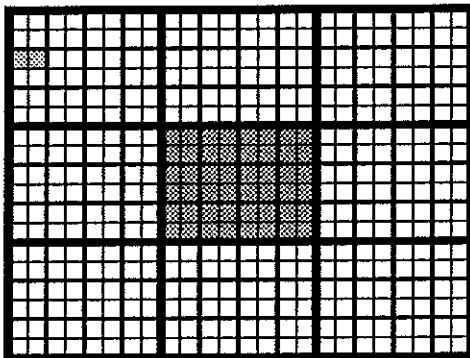
Positive coverage = 13

Negative coverage = 0

Best rules:

$[H=S][Ho=B]$

$\vee [H=R][B=S][S=Y][Ho=S][J=R][T=N]$



Iteration 11

Cost = 11 : 2 rules, 7 conditions

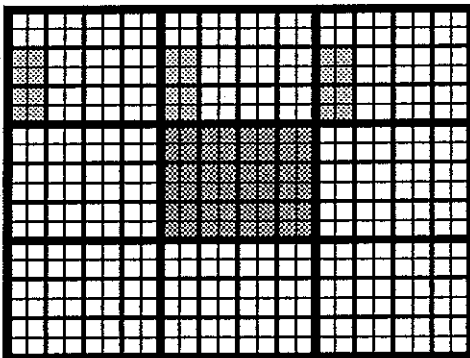
Positive coverage = 14

Negative coverage = 0

Best rules:

$[H=S][Ho=B]$

$\vee [H=R][B=S][S=Y][Ho=S][J=R]$



Iteration 16

Cost = 9 : 2 rules, 5 conditions

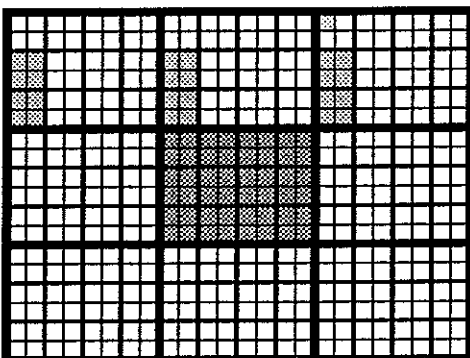
Positive coverage = 16

Negative coverage = 0

Best rules:

$[H=S][Ho=B]$

$\vee [H=R][B=S,O][J=R]$



Iteration 25

Cost = 17 : 3 rules, 11 conditions

Positive coverage = 17

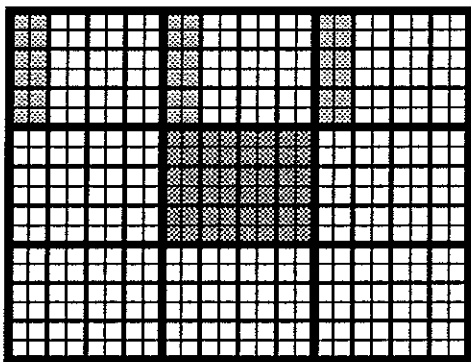
Negative coverage = 0

Best rules:

$[H=S][Ho=B]$

$\vee [H=R][B=S,O][J=R]$

$\vee [H=R][B=R][S=Y][Ho=F][J=R][T=Y]$



Iteration 27

Cost = 8 : 2 rules, 4 conditions

Positive coverage = 17

Negative coverage = 0

Best rules:

$[H=S][H_o=B]$

$\vee [H=R][J=R]$

This problem proved to be extremely simple to our algorithm. A complete and consistent description was found after only 27 basic iterations, and after 3.1 CPU seconds on a DEC3100 station. Moreover, this induction, based on the 20% of the available events, produced a concept description that was complete and consistent with all the unseen events (this is not necessarily the case with other learning systems — see table 9.1).

Chapter 8

AN ALTERNATIVE VIEW

In chapter 5 we described the ideas and the reasoning leading to the proposed design from the point of view of the GA community. In this section we show how the same design can be argued on the grounds of artificial intelligence and machine learning. We do that by exploring similarities between GAs and the most general problem solving paradigm of artificial intelligence — production systems, and by showing how the ideas behind the GA modifications can be explained in this framework.

The methodology of inductive learning, as defined by Michalski in [39], has been widely known and accepted. When restricted to attribute-based spaces, it can be shortly characterized as a rule-based framework with a set of inference rules and a set of principles. Why, then, does no symbolic system actually implement it directly? Among the known symbolic systems, only Michalski's AQ, with its beam-like search, at least implicitly follows the methods by performing simultaneous generalizations and specializations — by means of negations, unions, and intersections of formulas of the symbolic language. Quinlan's ID separates the two strategies completely: it builds complete and consistent descriptions by only specializing the formulas (starting with the most general one), and then it possibly applies generalizations by means of some tree pruning techniques. It seems that the reasons for such a diversion between the theory and practice are lack of clearly defined measures of partial achievements (heuristics) and the huge search spaces of inductive learning.

The space of possible concept descriptions in the attribute-based framework is extremely large. For example, using ten descriptive attributes with three values per domain we are faced with 3^{10} number of different rules. Then, the number of possible concept descriptions is $2^{3^{10}}$, or almost $10^{1000000000}$. Most approaches deal with this problem by performing only one kind of operation (*e.g.* specializations in decision trees), by searching a differently pre-enumerated space (*e.g.* neural networks), or by following a simplified strategy directed by some biased criteria (*e.g.* AQ). However, we want to use the inference rules directly in a rule-based framework. Then, one solution would be to use hill-climbing techniques. However, the available heuristics, relying mostly on partial measures of completeness and consistency, could easily lead to local traps. Therefore, irrevocable strategies are quite inapplicable. Another solution would be to use some tentative techniques. However, the huge search spaces would require such methods to be extremely well informed, or otherwise the database would

grow unmanageably fast. Again, the available heuristics are not strong enough to provide such qualities.

To solve this problem we use a search mechanism that keeps the database size under control by retaining only a fixed number of states. This is achieved by abandoning states that show little potentials, and fully exploring only those most promising. To implement this control we use the mechanisms of genetic algorithms, which exhibit exactly such behavior and were shown to be very robust. To organize the method, we use the ideas of *production systems*, whose general architecture is presented in figure 8.1. The database represents the state of the current search, the *production rules* (condition–action pairs) act on the database members to derive new states, and the control directs the rule–application process. A rule applied to the database alters its state.

Production systems were originally inspired by attempts to model the human cognitive process, and were first proposed by Post in 1943 ([59, pp. 28]). Their idea was to use production rules, which change states in a way of firing neurons. What artificial intelligence found interesting in production systems was the clear separation of various elements of the paradigm. This, in turn, allowed for transparency, modular design, and ease of both maintenance and knowledge refinement. Many different versions and generalizations of such a design were proposed by the artificial intelligence community, often specific for certain domains. However, they all share the ideas of the high level modularity, application of rule–like operators, and the name of production systems, AI production systems, production rule systems, *etc.* ([46]). Using such general views, one may argue that most expert systems are AI production systems, and so are some search methods, *e.g.* the best first search, when applied to specific domains.

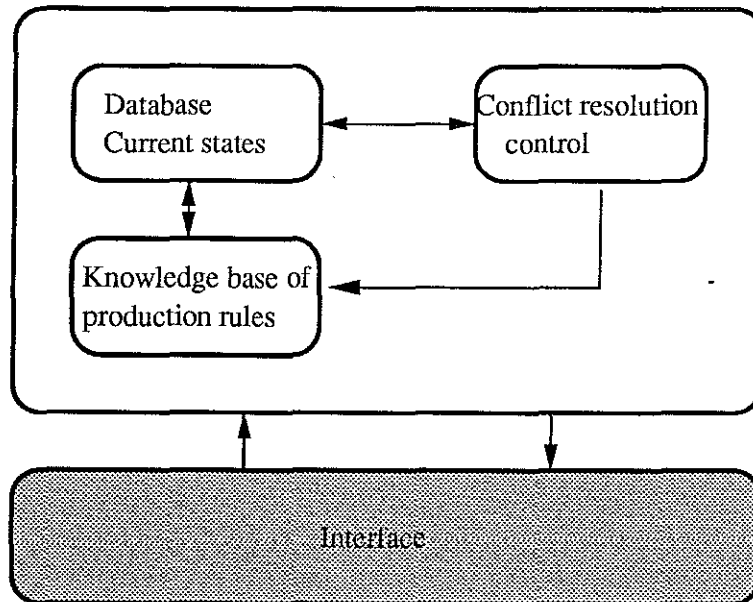


Figure 8.1: A production system architecture.

The three top level components are: a database representing the state of the current search (depending on the problem and the chosen representation, it might be either a single state reflecting all the transformations, or it might be a set of old and newly generated states), production rules representing the task-specific knowledge about state-transformations, and control for conflict resolution between candidate rules (see figure 8.1). The system starts either with the state reflecting the initial situation (*forward direction*) or with the desired goal (*backward direction*). In either case, the system runs until an appropriate sequence of rules transforms the initial state to the opposite one (*e.g.* to the goal after starting with the premises). Usually both the initial and the goal states are known a priori. In this case the solution is the path that transforms one into the other. In general, the ultimate goal may be to find the state that best satisfies some criteria. In this case the path may be of no interest.

A rule can change a state if its condition part is satisfied (the rule can *fire*). However, at a given moment a number of rules may candidate for firing. It is the task of the control to decide which rule actually alters the state. With this process, the database often rapidly grows in size, causing a chain reaction by increasing the number of candidate rules. Then, the feasibility of such a system depends on the control's quality. However, the control relies on heuristics for its decision making. Therefore, weak heuristics prohibit production system architectures from use in many large search spaces. In particular, this is the reason that such an architecture has not been applied to the task of supervised learning, despite the well identified representation and the inference rules of the inductive methodology. Before we further describe our solution, let us compare the GAs and production system architectures.

Looking back at the general architecture of a GA (figure 5.2), it seems these two are very similar. Both separate the current situation (database or population), the operators acting on such structures, and the control mechanism that directs the expansion process. However, there are a number of significant differences as well. In production systems, this separation is a natural by-product of the high abstractive level; on the other hand, genetic algorithms make this distinction based on different principles, often on a level that is distant from the conceptual level of the problem. Production rules become candidates for applications in a deterministic manner, by satisfying their conditions; operators of a genetic algorithm do not have any dependencies on the current situation, but rather act nondeterministically. In production systems the size of the database either grows or stays as just one growing state, depending on the choice of the representation; in genetic algorithms the population holds a constant number of states — to accommodate new states some of the stochastically worse chromosomes are abandoned. In production systems only one rule fires before candidacies is reconsidered; in genetic algorithms all selected rules fire in a basic cycle, independently of each other. In production systems a rule always applies to a single state to generate a new state; in genetic algorithms a new state may incorporate partial information from two parent states. Finally, in production systems a new state is a product of a new state and exactly one inference, while in genetic algorithms a new chromosome may be a product of few operations.

The new design of chapter 5 seems to push the genetic algorithm closer to the production systems: the inference rules still have the non-deterministic element, but

also have a deterministic, problem-specific aspect (*i.e.* some operators require simple condition matching, and probabilities of applications are adjusted by some problem-specific characteristics, namely the current coverage and the average chromosome size); the operators encode high level knowledge about state-transformations; different initializations provide a kind of bi-directional behavior. Then, the design itself can be described as a rule-based architecture, with the rules partially nondeterministic and modeling the task-specific knowledge, with a constant number of expanded states, and with the control following that of genetic algorithms. Since such mechanisms are known for their robustness, the system should operate in the immense space under the constraint of those weak heuristics.

The system can be pushed even closer to production systems, and we hope that exploring such possibilities can further improve both its quality and time performance. For example, initial experiments indicate that inverting the application mechanism from operator-oriented to state-oriented can improve the system's speed by a factor of five, all without a noticeable change in its quality. In addition, exploring such relations can lead to designs of new general artificial intelligence solving paradigms utilizing the search principles of genetic algorithms.

Finally, after these preliminaries, we are ready to describe the system in terms of the three components of rule-based systems: the database, inference rules, and control.

8.1 Database

We use the rule-based framework of the VL_1 language. Consequently, the search space becomes the space of all VL_1 descriptions. The goal is to find the best state that fits some criteria (completeness, consistency, and possibly some learning bias), and not the path leading to such a state. Then, each state of the current database is a sets of VL_1 complexes: a potential solution. Each complex is a conjunction of conditions that must be satisfied. Finally, each condition is related to exactly one attribute, and is equivalent to the VL_1 selector.

8.2 Operators

The operators transform states of the database to new (possibly better) states in the search space. Since the system operates in the space of VL_1 -based descriptions, the operators model those of the inductive learning methodology when restricted to the attribute-based spaces. To fully use the idea of the population, and to provide similar qualities as those of the genetic algorithm search (*i.e.* robustness), we introduce an additional operator which exchanges pieces of information between different states ("rule exchange"). We also define few additional operators utilizing some additional ideas of inductive learning, as replacing a number of rules by their most specific generalization, or by their most general specialization. Having the advantage of knowing the AQ and ID algorithms, we define two operators simulating the basic steps of those

two systems. Thus, we have an operator which, given a rule and a negative inconsistent event, replaces this rule by all maximal rules consistent with the old rule and that event ("rule directed split"). Following the ideas of the ID algorithm, we have an operator which partitions a rule using exactly one attribute ("rule split"). Finally, we also have a rule which, given a state (*i.e.* a rule set) and a positive uncovered event, adds this event as a new rule to this current description ("new event"). Additional heuristics are used to match operators to specific properties of intermediate descriptions (by dynamically adjusting the application probabilities).

Each operator is given an initial probability of application to its type of structure (based on the level of its definition: set of complexes, complex, and selector levels). These probabilities have the dynamic character with respect to the current context, *i.e.* to the current completeness and consistency. Prior probabilities of generalizing operators are increased for applications to structures (rule sets, rules, conditions) that are incomplete. Prior probabilities of specializing operators are increased for applications to structures that are inconsistent. Moreover, the levels of probability increase/decrease are based on the levels of inconsistency/incompleteness. In other words, these two measures serve as heuristics guiding the selection of appropriate operators.

8.3 Control

The control follows that of genetic algorithms. First, the initial database is filled with random rules and possibly some positive training events. Then, all the rule sets are evaluated to provide some heuristic measures of accomplishments. Such measures use the idea of normalized correctness based on completeness and consistency. To prevent redundancy in the descriptions, the measure is slightly adjusted with respect to the number of complexes in a description (however, we do not use any redundancy removing axioms). In addition, one may want to include an additional bias based on some learning criteria. For example, in the experiments shown here we also accommodate a complexity measure (as in [69]) which reflects the number of conditions.

Following these preliminaries, the algorithm enters a stage in which the three iterative steps are performed (selection, reproduction, evaluation) until a desired state is found. Since, in general, we do not know the sought description a priori, the termination condition may be based on the amount of resources available or some criteria for the solution. During selection a new database is generated by choosing stochastically better states (with repetitions). In other words, the most promising states may appear multiple number of times in the new database, while the weak ones may be abandoned. During reproduction the set of inference rules hypothetically applies to all structures of the newly selected database. Some of the operators require a simple condition to be satisfied in order to candidate for firing, others do not have any preconditions. However, the actual firing is non-deterministic, with probabilities based on prior values and the context in which each operator applies (completeness and consistency of current structures). All operators found to fire perform their actions: there is no conflict resolution. This action generates a set of new states,

which are offspring of the higher-valued previous states. Finally, all new states are reevaluated using the available heuristics.

8.4 Efficiency Considerations

Our operators are defined as simple atomic actions, which can be performed very efficiently on the VL_1 descriptions if a special representation is used. However, some of the operators require finding a proper positive or negative event. Moreover, the reevaluation of a state requires an extensive pattern matching against all training events in order to estimate its completeness and consistency. This may be uneconomical and highly inefficient. To deal with this problem we precompile the training data into binary coverage vectors, and we subsequently operate on such structures.

Chapter 9

EXPERIMENTS

In this chapter we report the results of an extensive testing of our approach (using the GIL implementation), aimed at both evaluating its behavior under various conditions and comparing its performance to that of other well known systems. All of the forthcoming results were obtained with the same implementation options and parameters as those of chapter 7. The only difference was an increase in population size to 50, and a change in the number of iterations as indicated individually.

9.1 Experimental Methodology

In the literature, systems are being evaluated and compared on the basis of quality while working with a standard set of well recognized artificial and real data: examples widely used are random DNFs, multiplexers, soybean disease, breast cancer. To evaluate quality of our GIL system we use some of these standard data sets. This also allows us to use published results obtained while experimenting with other systems, under the assumption of repeating exactly the same experimental sessions.

As mentioned in section 3.4, the acquired knowledge must meet two criteria: high predictive accuracy (classification) of unseen events and comprehensiveness at some high cognitive/conceptual level. The quality of the former measures the generalization power of the system. Since the recognition is normally done by associating numerical weights of confidence in different decisions, we call it a *quantitative* property. On the other hand, we call the latter a *qualitative* property.

The most common experimenting methodology is to split the available events into training and testing groups (usually 70% and 30%). Subsequently, the experiment calls for a learning session using the training group, followed by testing using the other group (containing events unseen during the training). Different measures are then used to determine the qualities of a system. For example, for the quantitative properties, Michalski and Chilausky define a set of conflicting measures ([38]), under the assumption that in some cases it makes no sense to distinguish between two close diagnoses. However, most researchers use a single measure of accuracy, defined as the ratio of correctly classified events to all testing events: this is the measure we use unless otherwise stated. To measure the qualitative properties we either list

separately the number of rules and conditions used, or combine them according to the well accepted formula shown in section 5.4.

Another important issue is the estimation of such assumed measures. If we had an infinite number of training and testing events, we would not have to worry about sampling errors (but would have to worry about time complexity). Having only limited resources available, the simplest way to estimate the measures is to run a single session. However, the confidence in such results is very low. Such experiments are permissible only under extreme conditions (*e.g.* time constraints). A more powerful method is to repeat the experiments with randomly selected training and testing groups, and report the average results. This is the most often used scenario. Under certain conditions (small number of available data events and low time complexity of the learning/testing session), a preferred method is so called *leaving-one-out*. This approach uses all-but-one events for training and the remaining event for testing, repeating the process for all available events and reporting the average results. In designing our experiments we use the second of the three described methods, with five random resamplings per test (unless otherwise stated).

9.2 Emerald's Robot World

Recently, a report of the AI laboratory at GMU was published ([69]) evaluating a number of different learning systems using the the world of robots from the Emerald system ([29]). Since this report provides a detailed description of the experiment, it is relatively easy to repeat exactly the same tests with our system and compare the results directly with those reported.

The following were the six variables describing the robot world:

Attribute	Values
<i>Head</i>	<i>Round, Square, Octagon</i>
<i>Body</i>	<i>Round, Square, Octagon</i>
<i>Smiling</i>	<i>Yes, No</i>
<i>Holding</i>	<i>Sword, Balloon, Flag</i>
<i>Color</i>	<i>Red, Yellow, Green, Blue</i>
<i>Tie</i>	<i>Yes, No</i>

and the robots were classified into the following five categories (created by a task-unaware human):

Concept	Description
C_1	<i>Head is Round and Color is Red or Head is Square and Holding a Balloon</i>
C_2	<i>Smiling and Holding a Balloon or Head is Round</i>
C_3	<i>Smiling and not Holding a Sword</i>
C_4	<i>Jacket is Red and no Tie or Head is Round and is Smiling</i>
C_5	<i>Smiling and Holding either a Balloon or a Sword</i>

Finally, the task was to learn a description of each concept while seeing only a varying percentage of the positive and the negative examples. There were a total of 432 different robots present in this world. The error rate reported is the average error in recognizing all the 432 (both seen and unseen) events. This measure explicitly estimates a system's predictive accuracy, while at the same time implicitly judges the system's generalization and specialization power.

System	Learning Scenario (Positive%/Negative%)				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	22.8%	5.0 %	4.8 %	1.2%	0.0%
BpNet	9.7%	6.3%	4.7%	7.8%	4.8%
C4.5	9.7%	8.3%	11.3%	2.5%	1.6%
CFS	21.3%	20.3%	21.5 %	19.7%	23.0%
GIL	4.3%	1.1%	0.0%	0.0%	0.0%

Table 9.1: Error rate summary in the robot world.

Concept	Learning Scenario (Positive%/Negative%)				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
C_1	11.1%	5.3%	0.0%	0.0%	0.0%
C_2	0.0%	0.0%	0.0%	0.0%	0.0%
C_3	0.0%	0.0%	0.0%	0.0%	0.0%
C_4	10.4%	0.0%	0.0%	0.0%	0.0%
C_5	0.0%	0.0%	0.0%	0.0%	0.0%

Table 9.2: GIL's error rate in the robot world.

The systems used in the mentioned experiment were: rule-based AQ15, neural network BpNet, decision tree with rules generator C4.5, and genetic classifier system CFS (for details see [69]). Table 9.1 reports the average error rate for the five experimental concepts for all five systems (the results of the other four obtained from those published experiments). Surprisingly, GIL (run here for the same 100 iterations) produced the highest recognition rate, especially when seeing only a small percentage of the events. Its concept-by-concept results are presented in table 9.2. It is interesting to note that the two most difficult concepts (C_1 and C_4) had the most uneven division between the number of positive and negative examples, causing a problem when learning with an insufficient number of events (*e.g.* 6% of the positive instances of C_1 were only about five objects).

Table 9.3 reports the average acquired knowledge complexity by listing both the average number of rules and the average number of conditions, as learned by all five systems for different learning scenarios. The NR entry indicates that the complexity

System	Learning Scenario (Positive%/Negative%)				
	6%/3%	25%/10%	50%/10%	75%/10%	100%/10%
AQ15	2.6/4	1.6/3	1.6/3	1.6/3	1.6/3
BpNet	NR	18/29	NR	NR	32/54
C4.5	6.8/12.2	4.4/9.2	4.8/9.2	4.8/9.2	3.8/7.3
CFS	NR	NR	NR	NR	NR
GIL	1.4/2.6	1.6/3	1.6/3	1.6/3	1.6/3

Table 9.3: Complexity's summary in the robot world.

Concept	Learning Scenario (Positive%/Negative%)				
	6%/3%	25%/10%	50%/10%	75%/10%	100%/10%
C_1	2/4	2/4	2/4	2/4	2/4
C_2	2/3	2/3	2/3	2/3	2/3
C_3	1/2	1/2	1/2	1/2	1/2
C_4	1/2	2/4	2/4	2/4	2/4
C_5	1/2	1/2	1/2	1/2	1/2

Table 9.4: GIL's complexity in the robot world.

was large and not reported in the reference paper. The reason for the higher complexity of the connectionist approach is that this is a non-symbolic system operating on numerical weights rather than on the problem symbols. On the other hand, the high complexity of the genetic approach can be attributed to the fact that the symbolic processing was being done in the representation rather than the problem space. This result is rather common for genetic algorithm approaches. Therefore, it was a big surprise to find that GIL's knowledge was at the same complexity level as that of the highly acclaimed AQ15. Again, GIL's results on a concept-by-concept basis are presented in Table 9.4.

Finally, for the most difficult learning concept C_1 , and for the most difficult reported learning scenario (6% positive and 3% negative events for training), we present a diagrammatic visualization ([69]) of the acquired knowledge as compared to the target concept (figure 9.1). Among the systems not shown, C4.5 produced a slightly more fragmented description, while the remaining two produced descriptions quite conceptually unrelated to the target (see [69]).

9.3 DNF Concepts

Learning DNF descriptions has become a standard way of evaluating different systems (*e.g.* [25]). An interesting experiment was conducted by Spears and DeJong ([13]),

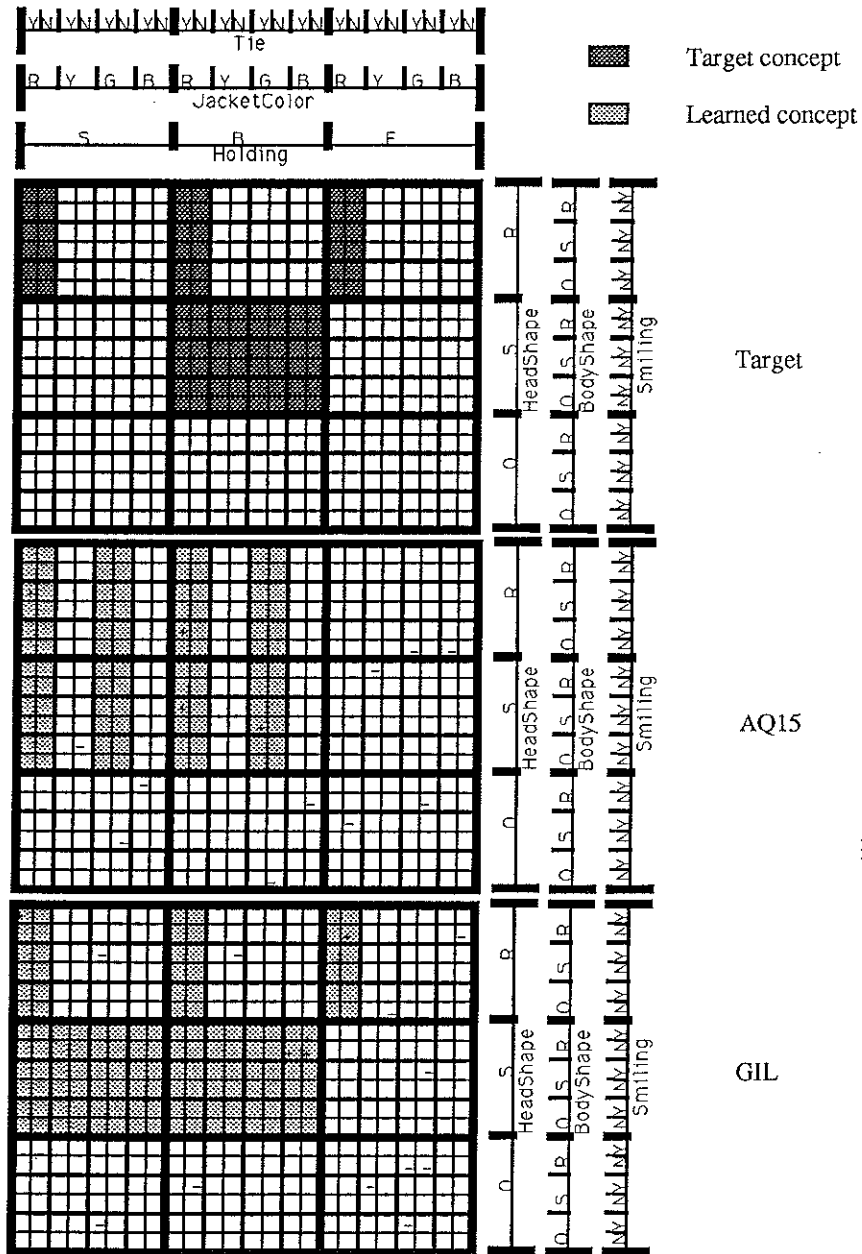


Figure 9.1: Diagrammatic visualization of the acquired knowledge.

in which they compared decision tree based ID5 with their own genetic algorithm for supervised concept learning GABIL. The test data for that experiment was a set of random DNF descriptions of a varying complexity. The reported results represent batch-incremental learning curves: a system's quality measure after seeing n examples is defined as an average recognition of a single unknown random event over the last ten experiments (from $n - 9$ to n). Accordingly, the learning curves are undefined for $n < 10$.

There were a total of six attributes, each having three possible values. Six sets

of experiments were conducted, for six randomly constructed DNF concepts of the following kind:

Concept	#Rules	#Conditions/Rule
1d1c	1	1
2d1c	2	1
1d2c	1	2
2d2c	2	2
1d3c	1	3
2d3c	2	3

For each experiment, a total of 100 events was chosen randomly. Then, using an increasing number of learning events and just one testing event, the learning curves were constructed using average results over ten independent runs with resampling. For the incremental ID5, the knowledge was updated incrementally upon a new inconsistent event, while it was generated from scratch in the GABIL system (it does not possess such incremental properties).

We repeated the same experiments in exactly the same environment, using the same batch-incremental mode as GABIL. Because the DNFs actually used were not reported in that paper, we repeated the experiments not only with resampling, but also with randomly regenerated target descriptions (each run of 100 iterations). The results are presented in figure 9.2. The original claim of the GABIL system was that it could not learn as well as ID5 on simple concepts, but achieved about the same levels (in some cases slightly better) of performance as the concepts' complexity increased. Our results show that GIL can do both at the same time: it achieves very high performance for all kinds of problems. Moreover, it clearly outperforms GABIL in terms of learning variability: its smooth learning curve indicates low variability in performance, while the broken curve of GABIL indicates bigger differences from run to run.

9.4 Multiplexers

The family of multiplexers is another widely used set of data. Each multiplexer is actually a specific case of the more general DNF. For each integer $k = 1, 2, \dots$ there is a multiplexer boolean function defined in the following way: the function's inputs are the k bits (called addresses), and there are exactly 2^k outputs (called data bits). Accordingly, we have multiplexer f_3 for $k = 1$, f_6 for $k = 2$, f_{11} for $k = 3$, etc. . The function of a multiplexer is to activate the data bit whose address (in binary, assuming some ordering of the data bits starting at 0) is specified by the address bits. For example,

$$[A_0 = 0][A_1 = 0][A_2 = 1] \vee [A_0 = 0][A_1 = 1][A_3 = 1] \\ \vee [A_0 = 1][A_1 = 0][A_4 = 1] \vee [A_0 = 1][A_1 = 1][A_5 = 1]$$

defines the f_6 multiplexer in the VL_1 language, assuming that attributes A_0 and A_1 are the two address bits, and A_2 to A_5 are the four data bits.

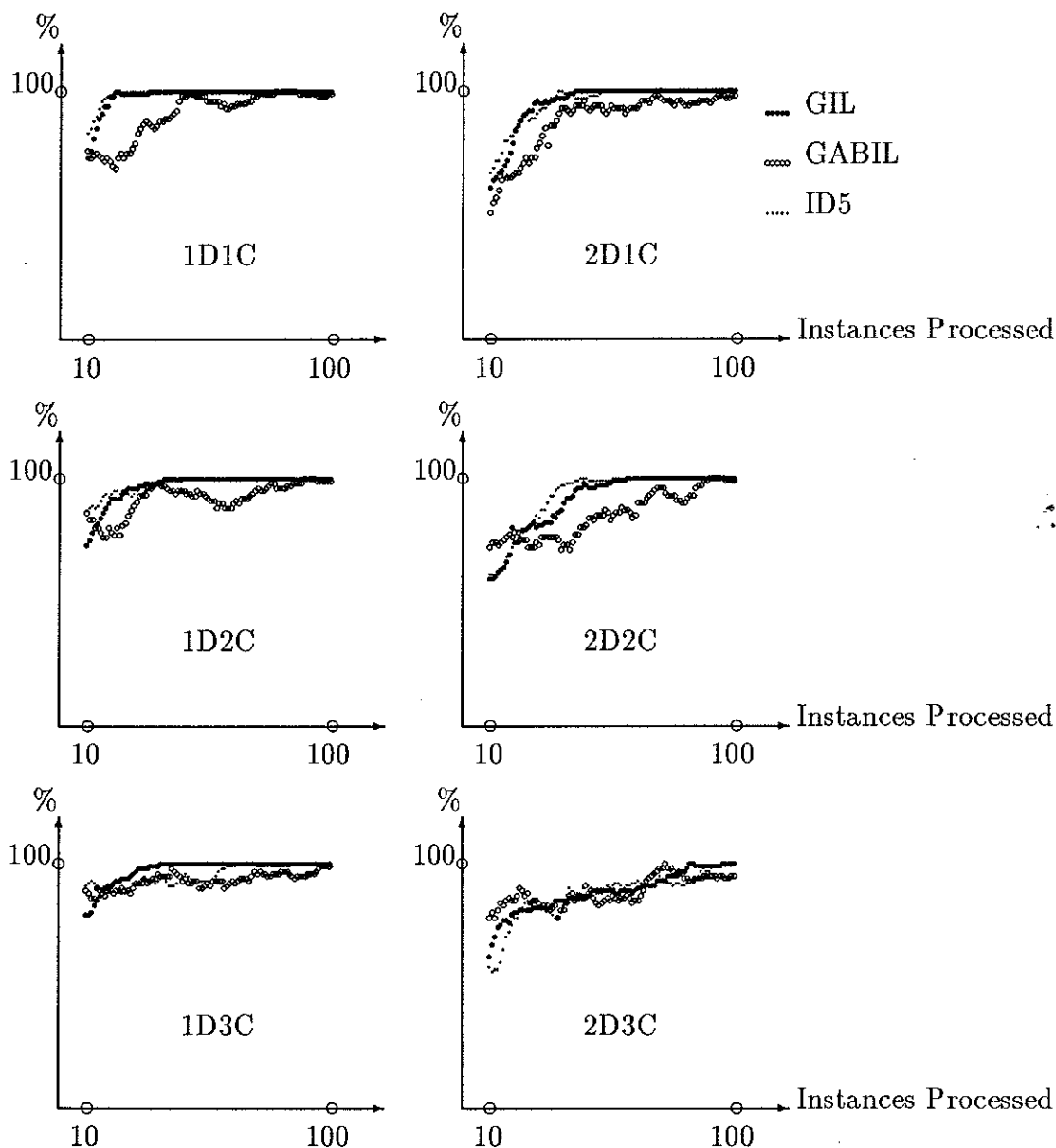


Figure 9.2: Batch-incremental results on DNF data.

Many experiments have been documented, mostly using f_6 and f_{11} functions. For example, Koza describes learning the first of these two, while using his LISP-influenced hierarchical genetic approach. He reports a case of learning the actual function after processing 4500 potential solutions, while seeing all 64 (2^6) possible instances. Our own experiments indicate an average learning after seeing only about 2000 individuals (40 iterations, database size 50). However, these two results should not be compared directly — we lacked the necessary details to recreate the same setting. We performed a different experiment while presenting our system with an increasing number of training events. The resulting learning curve is presented in figure 9.3. In all these experiments, GIL ran for 200 iterations, which took an average

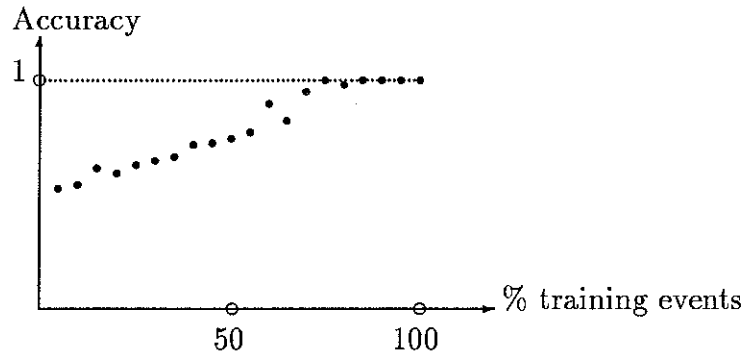


Figure 9.3: Learning curve on multiplexer f_6 .

of 8.5 CPU seconds on a DEC3100 station.

Multiplexer f_{11} is considerably more complex than the simple f_6 ; the size of its event space is 2048, or 2^{11} . Table 9.5 reports an average accuracy while learning with varying sizes of the training set. These results are similar to those from other systems (*e.g.* [52]), but a different experimental methodology does not allow for a direct comparison. An interesting fact observed here was that this learning was quite slower than that of f_6 — an average of 20 CPU minutes on the same DEC3100 station for 2000 iterations. There seem to be two major reasons for this increase. Firstly, the complexity of the sought here description was 48 vs. 20 in the previous case. Secondly, the number of possible complexes is much larger in the second case, giving a much larger search space: approximately 10^{60000} vs. 10^{200} . Indirectly, the larger search space required more iterations for the learning, and larger intermediate states (see figure 9.4) caused a longer processing on each iteration. Nevertheless, this complexity still compares very favorably with other GA approaches. We hope to improve this even further by a more efficient selection mechanism and a parameter tuning (we return to these issues in chapter 10).

% training events	Accuracy
5%	77%
10%	88%
20%	94%

Table 9.5: GIL's accuracy on multiplexer f_{11} .

Yet another interesting difference between the two multiplexers can be observed by comparing the accuracy while learning with a small percentage of the available events: f_{11} achieves high rates much quicker. The reason for such a behavior seems to be the ratio of the concept complexity to the size of event space, which is about 0.313 for f_6 and 0.023 for f_{11} . This raises an interesting hypothesis about estimating the difficulty of generating descriptions: such a difficulty is proportional, or at least highly correlated, to the ratio of the concept's complexity and the size of the event

space. We hope to further investigate this assertion in the future.

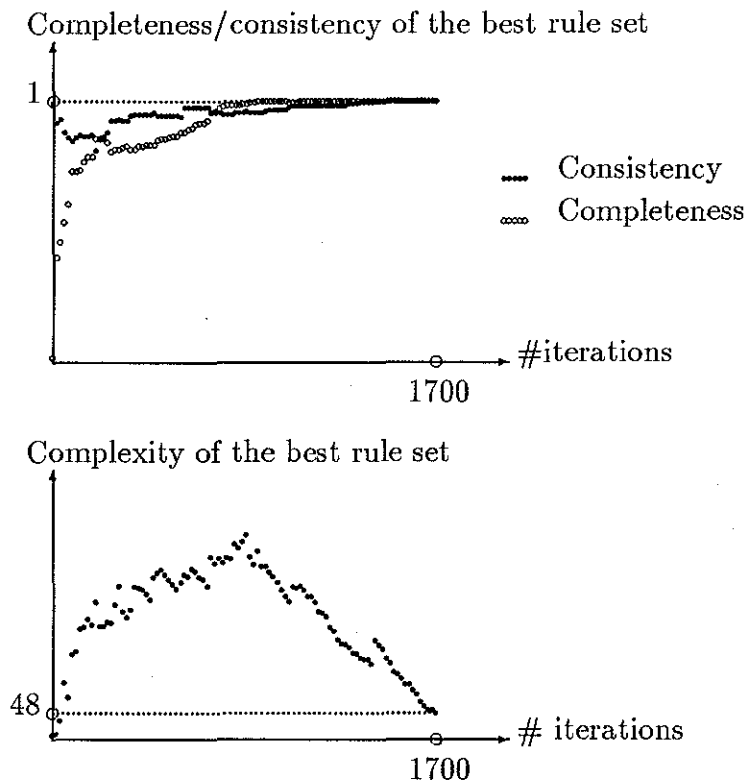


Figure 9.4: A sample behavior on multiplexer f_{11} .

Figure 9.4 traces a sample run while training with 20% of the available events. During this learning session, the exact concept was learned after 1700 iterations. A consistent and complete description of the training events was found shortly after 1000 iterations, and the remaining 700 cycles were required to simplify the generated description. The first of these two graphs traces completeness and consistency of the currently best database individual at 17-iterations intervals (100 data points). The other graph traces the complexity of such best individuals. It is interesting to note that the complexity rises during the learning, as a result of not finding simple enough complete and consistent descriptions, and then decreases — forced down by an increasing cost influence (see section 5.4) and formation of such better descriptions.

9.5 Breast Cancer

The breast cancer data is one of the most popular natural domains used in experiments with inductive learning systems. It contains 286 descriptions of female patients, classified as either developing or not developing a recurrence of breast cancer after a five year period following the first surgery. The descriptions are generated using nine attributes, with an average of 5.8 values per domain. Such descriptive language was found to be inconsistent, meaning that some patients having exactly the same

description were classified differently. Such a situation puts an extra burden on the learning system.

System/method	Complexity	Accuracy
Human experts	Not reported	64%
AQ15/full rule set	41 rules/160 conditions	66%
AQ15/best rule only	2 rules/7 conditions	68%
Assistant/without tree pruning	63 leaves/120 nodes	67%
Assistant/with tree pruning	9 leaves/16 nodes	72%
GIL	36 rules/128 conditions	65%
GIL/with emphasized cost	10 rules/27 conditions	67%

Table 9.6: Summary of the breast cancer experiment.

An excellent publication by Michalski and colleagues ([41]) lists both quantitative and qualitative results on this data set, while using the AQ15 system with a rule truncation mechanism and a decision tree system ASSISTANT (this version generates low complexity binary trees and employs a tree truncation technique: [31]). In addition, this publication also reports the accuracy of human experts. We repeated these experiments (for 1000 iterations) and report all such results in table 9.6¹. They indicate the applicability of our approach in the case of natural domains as well. For compatibility with the complexity results reported for the other systems, we ran GIL twice, each time learning one the two possible concepts, and summarizing the #conditions and #rules.

9.6 Dealing with Empty and Invalid Rules

These experiments study the effect of different treatments of invalid and empty rules. An invalid rule occurs if one of its conditions becomes fully restricted, *i.e.* excludes all domain values. Preserving such rules increases the memory contents of the system. This has two effects:

- Useful blocks of information (*e.g.* partial rules) are not immediately dispensed upon an overspecialization, but are preserved for future reference (and possible back-generalization).
- The memory size (database size) of the system increases, causing slower performance.

A similar dual effect occurs in the context of empty rules. An empty rule is that not covering any positive events. Intuitively, such rules might also be very useful in exploring the search space, but they again increase the size of the database when retained.

¹Database courtesy of the AI Center, GMU.

Which rules retained	CPU seconds	Accuracy	#rules
None	25.4	0.90	4.7
Invalid	26.4	0.94	4.9
Empty	28.1	0.92	5.0
Both	49.7	0.95	5.8

Table 9.7: The effects of retaining invalid and empty rules.

Experiments confirm this anticipated trade-off. Table 9.7 compares run times, predictive accuracy, and the average size of an individual of the database over the life of the simulation, for four different cases: retaining both kind of rules, deleting only each kind of rules separately, and deleting both. It can be observed that increasing the memory size, by retaining any of these two kinds of rules separately, increases the system's accuracy, while also increasing the time complexity. In both cases, the relatively small complexity overhead is well compensated for by the increase in recognition quality. Retaining the empty rules seems to have a smaller effect on this quality, but this result may be well related to the high percentage of the search space events used for training. Finally, retaining both kinds of rules simultaneously produces the best accuracy increase. However, this is associated with a relatively high increase in time performance. Such a high increase was rather unanticipated, and we suppose it may be related to the implementation of the system itself. More tests are needed to test this phenomenon.

These experiments were conducted using the f_6 multiplexer, 70% events for training, and 500 iterations. The number of iterations was fixed as means of generating comparable runs, even though in many instances the correct concept was found much faster. All results are averages of ten resampled runs.

9.7 Incremental Learning

In section 3.5 we mentioned that incremental learning capabilities are important attributes of a learning system. This is the case mainly for two reasons: human learning shows incremental characteristics, and a natural setting for a learning system often displays dynamic properties — new evidence becomes available from time to time, as might be the case in a medical database environment where some of the diagnoses may be confirmed after a while, or when new patients are diagnosed. Theoretically, the incremental character is obtained by abilities to generalize and specialize existing knowledge, as necessary, upon new experience. We would expect our approach to possess such properties naturally, as it is based on generalization and specialization of the current hypotheses, and it does not explicitly favor any of these two classes of actions at any time. To evaluate this assertion, we repeated the batch mode experiments with the f_6 multiplexer, but in an incremental setting: learning was performed using the available data set and 100 generations, with the accuracy estimated. Then,

new training events were added (and the old ones remembered as well — full memory learning) to that set in 5%–increments, and the system would be allowed to learn again. Each time the initial database was set to the final database of the previous learning session, the previously generated knowledge.

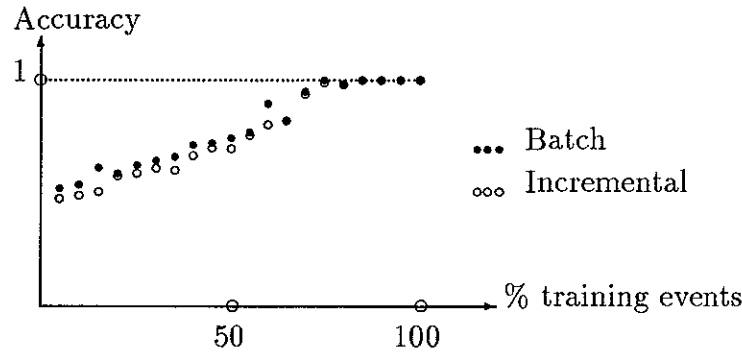


Figure 9.5: Comparison of the batch and incremental learning.

The results (figure 9.5) indicate that for a small number of training events the incremental mode generates descriptions of slightly lower accuracy, but the difference vanishes for a higher percentage of such training events. This behavior can be explained by observing that a smaller percentage of training events generates lower accuracy knowledge in general, which is further from the true descriptions and highly biased by the choice of the initial training events. As such a percentage increases, the current knowledge gets closer to the sought descriptions, and the extra training events become more consistent and less influential.

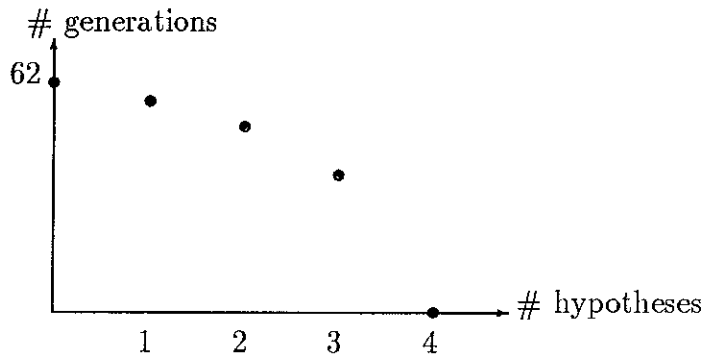


Figure 9.6: The effect of initial hypotheses in the initial database.

9.8 Initialization

In section 5.3 we described three different components of the initial population our system can accommodate: random rule sets, positive events, and initial hypotheses. We also mentioned that the actual importance of filling the population with positive events diminishes with the introduction of the “new event” operator, which

inserts uncovered positive events to chromosomes. In this section we study the system's ability to accommodate background knowledge in form of initial hypotheses. In our experiments we used f_6 multiplexer with all 64 events available for training, and we measured the number of iterations needed to find a complete and consistent description. Population size was set to 50.

Figure 9.6 presents the results obtained: the number of generations needed for learning the exact concept description, as a function of the number of sought and uncorrupted disjuncts presented as input (out of four). A similar pattern exists if the hypotheses represent somehow corrupted partial descriptions. The results indicate that the system is capable of using such extra information to improve its performance, justifying its potential applicability as a dynamic knowledge refinement tool.

Chapter 10

CONCLUSIONS AND FURTHER RESEARCH

We have described a novel approach to supervised inductive learning in attribute-based spaces which uses a knowledge-intensive genetic algorithm. Such an algorithm follows the ideas of traditional GAs, but replaces the domain-independent search by domain-specific inference operators modeled upon those of the inductive learning methodology. This approach represents an abstraction of the traditional genetic algorithm to the symbolic level. Initial results show that GAs can be successfully applied to more complex, non-numerical, tasks by defining the algorithm at the conceptual level of the problem. This allows for processing high level structures using the problem specific methodology and rich heuristics. Moreover, such an abstract view provides for the same clear separation of different system's components as found in AI production systems. This modularity, in turn, allows for transparent applications of similar designs in other domains.

When pursuing this challenge we did not attempt to produce a system able to compete with the existing symbolic systems (AQ and ID based), especially in terms of time complexity. Our goal was rather to investigate the potentials of such a method of abstracting genetic algorithms, which may be carried to other domains. Nevertheless, by designing efficient data compilation methods aimed at reducing the system's complexity, and by using more "intelligent" operators than those in the traditional GAs, we were able to tackle a number of interesting problems in a reasonable time. This alone represents a large improvement over traditional GAs applications. Moreover, since genetic algorithms are naturally suited for parallel architectures, we may hope that such reimplementations, along with new technological advances, may be faster without of any additional efforts.

The system also shows significant potentials from the machine learning point of view. Firstly, it does not assume attribute independence, as the ID-based systems do. Secondly, it uses more problem specific heuristics (the inference rules) than the AQ-based systems do. Finally, it extends the AQ's ideas of exploring a number of simultaneous directions to a more powerful platform allowing for both competition and cooperation (by information exchange). Moreover, it should have potentially linear characteristics with respect to event space sizes, but a full explorations of

this assertion requires more extensive experimentations and an even more efficient implementation.

The current complexity, as well as the overall quality, seems to be unstable and to vary with a given problem. We hope that the parameter abstraction (section 10.1) can relax this dependency, but, again, it requires an extensive continuing research. Other important issues that should be also addressed in the future include learning multiple concepts (section 10.2), and dealing with noisy information — possibly by changing the rule-based conceptualization view to a more liberal one (section 10.3).

One should point out that most of the testing data was not suited to explore the full potential of this system: most experiments were conducted with two or three-valued domains. Such domains are much more suitable to the other learning systems while this approach can fully explore spaces with larger and typed domains. Actually, among the other known learning systems, only AQ15 tries to accommodate this extra knowledge, which can be quite valuable when the domains grow. However, it does so only after learning the initial complete and consistent descriptions. In this sense, this approach seems to be the first one to be able to accommodate typed domains for the benefit of learning.

An additional exceptional benefit of this research and its results arises in the context of inductive learning. Our results indicate the feasibility of the previously proposed inference operators of the inductive learning methodology. Moreover, our implementation presents a valuable tool to further studies on this subject, *e.g.* to determine the relative importance of different techniques, and their necessity. Moreover, we feel it is quite possible and feasible to extend this approach to more powerful languages, *i.e.* some extension of VL_1 .

10.1 Parameter Abstraction

One of the major disadvantages of the current implementation is its high parameterization: there are over 30 input parameters that must be specified, with most of them being continuous probability values. Under such conditions, it seems highly unlikely to select the proper combination for a given run. This is the reason for the worsened performance noticed on more complex problems (*e.g.* multiplexer f_{11}). Moreover, on some occasions we observed an improved performance after slightly changing the priori probabilities for a given problem. However, a much more extensive and systematic experimentation is necessary to determine the actual source of such variations: randomness of the runs or some problem-specific characteristics.

To deal with the problem of the size of the parameter space, it is necessary to explore inter- and intra-dependencies between such parameters. For example, all rule-set level application probabilities should be specified with respect to each other and, as a group, with respect to those of other groups. The dependence of the selection probabilities on the problem size should be explored. However, establishing such relations requires an extensive testing over a wide variety of different problems.

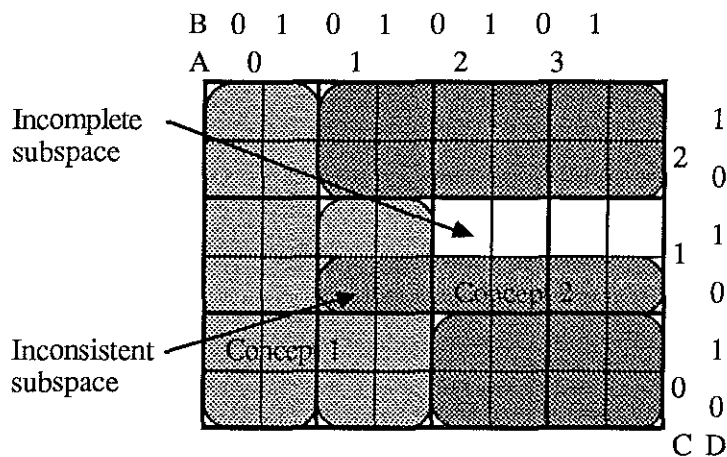
To deal with the second problem (dependency on characteristics of the problem), the choice of such abstracted parameters should be further (in addition to the dy-

dynamic method of section 5.5.2) associated with the problem complexity (determined dynamically in the process of learning) and some learning criteria (specified by the used or some other requirements). Then, all these parameters could be replaced by few conceptual ones, *e.g.* desired type of descriptions (as specific vs. general, or low attribute cost vs. low descriptive cost). Such an abstraction would provide for both an easier use and a more efficient performance.

10.2 Multiple Concepts

The current system assumes single-concepts only, and proceeds by learning only rules associated with the single decision. This lets us treat the decision implicitly, and subsequently simplify the definitions and implementation by processing VL_1 complexes in place of rules. For the sake of simplicity and continuity, we try to preserve this property while generalizing the approach to learning multiple decisions.

When discussing such generalizations, we must consider a related issue: treating descriptions that are incomplete and inconsistent with respect to the problem space. As we mentioned in section 3.3, rule-based framework is not well suited for learning descriptions with the above properties, and often these qualities are relaxed. However, in such a case, a previously unknown event may be recognized by none or multiple concepts. Then, a special arbitration protocol must be employed. The simplest such protocol simply returns such an event as unrecognizable: this improves correct recognition rate, but also increases overall indecision. Such an approach can treat descriptions of all different classes independently. A more sophisticated protocol employs some probabilistic measures, thus changes the conceptualization view to a non-crisp. However, this also suggests that the descriptions should not be generated independently, but in a common context. Accordingly, we propose two possible generalizations to learning multiple descriptions: one that assumes such rule sets independence, the other that does not.



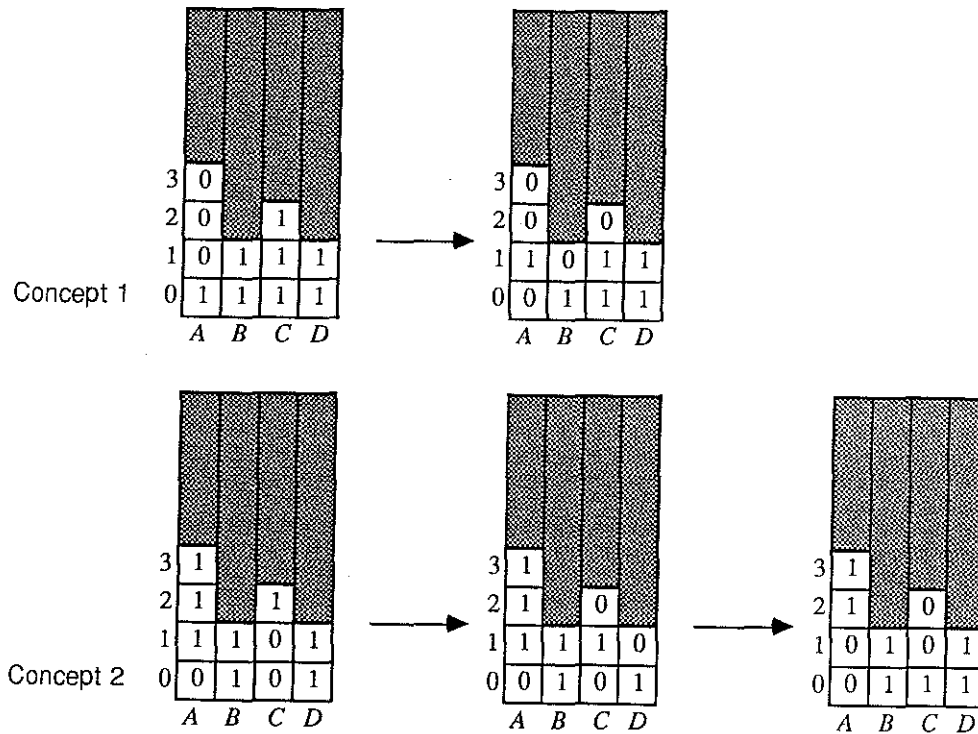


Figure 10.2: A sample internal representation of a two-concepts case.

may use the current system in subsequent learning sessions for each of the sought descriptions. Then, during a learning session for class n , all positive events of the category n are considered as positive events, while all positive events of all other categories as the current negative events. A possible result of such a learning scenario for two classes is presented in figure 10.1: some new events will not be classified because either the space is not covered, or it is covered in a conflicting manner. This simple extension was used in experiments of section 9.5. The same idea may be used differently: we may conduct the learning sessions simultaneously in different populations, with one population per class being learned.

The first generalization is impractical if we relax the description's independence assumption, for now each rule set must be evaluated in the context of descriptions for the other classes. Again, we can preserve the implicitness of decisions if we introduce an additional syntactic level: the concept set level. Then, while learning descriptions of n categories, each chromosome becomes a set of descriptions, with each description associated with exactly one of the categories. In other words, a chromosome becomes a set of the previously defined chromosomes (section 5.2). Then, all operators that were defined as acting on two chromosomes will act on two rule sets associated with the same decision. All other operators stay exactly as defined previously. However, we need new operators to act on the new (VL_1 set of sets of rules) level. The only defined operator has an independent character, and it exchanges one or more whole-category descriptions between two chromosomes. In other words, this operator works at the level of granularity of a whole rule set associated with one implicit decision.

As to the arbitration protocol for the second strategy, a very nice solution was proposed by Michalski ([42]) and used in the AQ family: the two-tiered conceptualization view. We suggest using the same approach in such an extended architecture, which could provide an additional advantage: the two-tiered view could be used during the learning process, while it is used only in a follow-up step in the AQ15 system.

Because a given problem specifies a priori the number of categories to be conceptualized, we can easily extend the chromosome implementation of section 6.2 to a vector of such, where a vector position is associated with the decision number. For example, figure 10.2 shows an internal representation of the two concepts of figure 10.1.

10.3 Noisy Information

While it might be conceptually advantageous to assume an existence of noise-free data, it is often unrealistic under a real world condition. Such noise is normally associated with both imperfect measurements (instrument and human errors), and with the need to represent some continuous domains by discrete sets of features. Therefore, an artificial system aimed to work in a realistic domain should deal with these issues. Some researchers addressed the problem of noisy features. For example, Clark ([7]) discusses the effect of noise on induction and presents his probabilistic way of dealing with the problem. Quinlan showed how to deal with the effect of noisy data in decision trees ([50]). The two-tiered representation of AQ family of systems applies to noise effect reduction as well (*e.g.* [70]). Looking at these attempts, it is fair to say that noise accommodation generally employs some kind of probabilistic approach. Our simplified approach assumes a crisp concept representation with rule-based conceptualization and is not well suited for dealing with noise: the most appealing approach would be to combine rule-coverage information with their complexity in such a way that light (very low positive coverage) rules become more costly and more applicable to be removed. The same strategy applies also to all of the proposed generalizations to deal with multiple concepts. However, in the case of the extended chromosome architecture, the two-tiered description may itself be more valuable as a noise-accommodation agent, as such descriptions were showed to improve noisy recognition in the AQ15 system.

10.4 Other Operators

We mentioned in section 5.5 that pursuing atomicity and efficiency over complexity made us neglect the inductive resolution rule. It would be an interesting study to compare behavior of the current system with another one implementing this operator, along with other possible new operators. Such a study should concentrate on both the quantitative and qualitative properties, as well as a possible quality vs. time trade off.

In addition, some of the rule set level operators described in section 5.5 could be differently defined, depending on the selection of applicable rules. For example, the “rules exchange” operator could overlook the selection probability and always select a single rule. On the other hand, the “rules copy”, “rules generalization”, “rules drop”, and “rules specialization” could be enhanced by such probabilities, instead of always choosing a fixed number of rules (one or two, depending on the nature of the operator). Again, an extensive study is required to establish values of such new operators in relation to those presently used.

10.5 More Intelligent Initialization

The ideas used here can be seen as a very specific case of more general ideas presented by Davis in recently published [8], where he calls for exploring combinations of the traditional domain-independent genetic algorithms with some known problem-specific methods (the hybrid approach). The only difference between this system and those ideas lies in the initialization: he argues to fill the first population by chromosomes representing properly represented results of some known fast systems. The argument for such an enhancement is that then the genetic algorithm can only improve such ad hoc solutions. Therefore, it is guaranteed to perform at least as well as such other systems. In our domain there are such programs. For example, most decision tree systems are very efficient and produce assertions easily convertible to a rule-based format. Since our initial experiments indicate the system’s ability to process some initial hypotheses, there is a potential for a significant performance and time improvement.

Bibliography

- [1] Antonisse, H.J. & Keller, K.S., "*Genetic Operators for High Level Knowledge Representation*", Proceedings of the Second International Conference on Genetic Algorithms, 1987.
- [2] Baim, P.W., "*Automated Acquisition of Decision Rules: the Problems of Attribute Construction and Selection*", Reports of the Department of CS, University of Illinois at Urbana-Champaign, 1984.
- [3] Baker, J.E., "*Reducing Bias and Inefficiency in the Selection Algorithm*", *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.
- [4] Bethke, A.D., "*Genetic Algorithms as Function Optimizers*", PhD Dissertation, University of Michigan, 1980.
- [5] Booker, Lashon B., "*Intelligent Behavior as an Adoption to the Task Environment*", Ph.D. Dissertation, University of Michigan, 1982.
- [6] Bosworth, J., Foo, N., Zeigler, B.P., "*Comparison of Genetic Algorithms with Conjugate Gradient Methods*", (CR-2093), Washington, DC: National Aeronautics and Space Administration.
- [7] Clark, P. and Niblett, T., "*Induction in Noisy Domains*", *Progress in Machine Learning*, Bratko, I. and Lavrac, N. (ed.), Sigma Press, 1987.
- [8] Davis, L. (ed), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.
- [9] DeJong, K.A., "*An Analysis of the Behavior of a Class Genetic Adaptive Systems*", Doctoral Dissertation, University of Pittsburgh, 1976.
- [10] DeJong, K.A., "*Genetic Algorithms: a 10 year Perspective*", *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [11] DeJong, K.A., *Genetic Algorithms: A 10 Year Perspective*, *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [12] DeJong, K.A., "*Learning with Genetic Algorithm: An Overview*", *Machine Learning* 3,1988.

- [13] DeJong K.A. & Spears, W.M., "Using Genetic Algorithms for Supervised Concept Learning", *Proceedings of the Second International Conference on Tools for AI*, 1990.
- [14] Davis, L., (editor), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.
- [15] Fisher, D.H., McKusic, K.B., "An Empirical Comparison of the ID3 and Back-propagation", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.
- [16] Forrest, S., "Implementing Semantic Networks Structures Using the Classifier System", *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [17] Greene, D.P. & Smith, S.F., "A Genetic System for Learning Models of Consumer Choice", *Proceedings of the Second International Conference on Genetic Algorithms*, 1985.
- [18] Grefenstette, J., "Incorporating Problem Specific Knowledge into Genetic Algorithms", *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.
- [19] Goldberg, D.E., "Genetic Algorithm and Rule Learning in Dynamic Control System", *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [20] Goldberg, D.E. & Holland, J.H., "Genetic Algorithms and Machine Learning", *Machine Learning* 3, 1988.
- [21] Goldberg, D.E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Co, 1989.
- [22] Holland, J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.
- [23] Holland, J.H., "Escaping Brittleness", *Machine Learning II*, ed. R. Michalski, J. Carbonell, T. Mitchel, Morgan Kaufmann, 1986.
- [24] Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.R., *Induction*, The MIT Press, 1986,
- [25] Sridharan, N.S. (ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.
- [26] Janikow, C.Z. & Michalewicz, Z., "A Specialized Genetic Algorithm for Numerical Optimization Problems", *Proceedings of the Second International Conference on Tools for AI*, 1990.

- [27] Janikow, C.Z., "An Experimental Study Comparing Symbolic and Subsymbolic Inductive Learning Systems", *Proceedings of FLAIRS-91: Machine Learning*, 1991, pp. 81-86.
- [28] Janikow, C.Z. & Michalewicz, Z., "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms", *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, to appear.
- [29] Kaufman, K.A., Michalski, R.S., Schultz, A.C., "EMERALD 1: An Integrated System of Machine Learning and Discovery Programs for Education and Research", Center for AI, GMU, User's Guide. No. MLI-89-12, 1989.
- [30] Kondratoff, I., *An Introduction to Machine Learning*, Pitman, 1988.
- [31] Kononenko, I., Bratko, I., Roskar, E., "Experiments in Automatic Learning of Medical Diagnostic Rules", Technical Report, J. Stefan Institute, Ljubljana, Yugoslavia, 1984.
- [32] Koza, J.R., "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs", *Proceedings of the International Joint Conference on Artificial Intelligence 1989*.
- [33] Lbov, G.S., "Selection of an Effective System of Dependent Features", Collection of papers of Inst. for Mathematics, Academy of Science, Novosibirsk, 1965.
- [34] Michalewicz, Z. & Janikow, C.Z., "GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints", *Communications of the ACM*, to appear.
- [35] Michalewicz, Z. & Janikow, C.Z., "Genetic Algorithms for Numerical Optimization", *Statistics and Computing*, to appear.
- [36] Michalewicz, Z. & Janikow, C.Z., "Handling Constraints in Genetic Algorithms", *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, to appear.
- [37] Michalski, R.S. & Larson, J.B., "Selection of Most Representative Training Examples and Incremental Generalization of VL Hypothesis", Report, Dept. of CS, University of Illinois at Urbana-Champaign, 1978.
- [38] Michalski, R.S. & Chilausky, R.L., "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 2, pp. 125-161, 1980.
- [39] Michalski, R.S., "Theory and Methodology of Inductive Learning", *Machine Learning I*, 1983.

- [40] Michalski, R.S. & Reinke, R.E. *"Incremental Learning of Concept Descriptions: A Method And Experimental Results"*, *Machine Intelligence 11*, 1985.
- [41] Michalski, R.S., Moztetic, I., Hong, J., Lavrac, N., *"The AQ15 Inductive Learning System: An Overview and Experiments"*, Report, Dept. of CS, University of Illinois at Urbana-Champaign, 1986.
- [42] Michalski, R.S., *"Two-Tiered Concept Meaning, Inferential Matching and Conceptual Cohesiveness"*, Dept of Computer Science, Univ. of Illinois, 1986.
- [43] Michalski, R.S., *"Understanding the Nature of Learning, Machine Learning II*, Morgan Kaufmann, 1986.
- [44] Michalski, R.S. & Watanabe, *"Constructive Closed-loop Learning: Fundamental Ideas and Examples"*, Technical Report, ML Center, GMU, 1988.
- [45] Michalski, R.S., *"Learning Flexible Concepts"*, *Machine Learning III*, Morgan Kaufmann, 1990.
- [46] Nilsson, N., *Principles of Artificial Intelligence*, Morgan Kaufmann, 1980.
- [47] Pearl, J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1989.
- [48] Quinlan, J.R., *"Learning Efficient Classification Procedures and their applications to Chess End Games"*, *Machine Learning I*, 1983.
- [49] Quinlan, J.R., *"Induction of Decision Trees"*, *Machine Learning*, Vol. 1, No. 1.
- [50] Quinlan, J.R., *"The Effect of Noise on Concept Learning"*, *Machine Learning II*, Morgan Kaufmann, 1986.
- [51] Quinlan, J.R., *"Generating Production Rules from Decision Trees"*, *Proceedings of the tenth International Joint Conference on Artificial Intelligence*, 1987.
- [52] Quinlan, J.R., *"An Empirical Comparison of Genetic and Decision-tree Classifiers"*, *Proceedings of the Fifth International Conference on Machine Learning*, 1988.
- [53] Rendell, L.A., *"Genetic Plans and the Probabilistic Learning System: Synthesis and Results"*, *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [54] Rendell, L., Cho, H., Seshu, R., *"Improving the design of Similarity-Based Rule-Learning Systems"*, *International Journal of Expert Systems*, Vol. 2, No. 1, 1989.
- [55] Rumelhart, D. & McClelland, J., *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, vol.1, MIT Press, 1986.

- [56] Schaffer, J.D., "*Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*", Ph.D. Dissertation, Vanderbilt University, 1984.
- [57] Schaffer, J.D., "*Learning Multiclass Pattern Discrimination*", *Proceedings of the First International Conference on Genetic Algorithms*, 1985.
- [58] Schaffer, J.D., "*An Adaptive Crossover Distribution Mechanism for Genetic Algorithms*", *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.
- [59] Schutzer, D, *Artificial Intelligence, An Application-oriented Approach*, Van Nostrand Reinhold Company, 1987.
- [60] Smith, E. & Medlin, D., *Categories and Concepts*, Harvard University Press, 1981.
- [61] Smith, S.F., "*A Learning System Based on Genetic Algorithms*", PhD Dissertation, Univ. of Pittsburgh, 1980.
- [62] Utgoff, P.E., "*ID5: An Incremental ID3*", *Proceedings of the 5th International Conference on Machine Learning*, 1988.
- [63] Vignaux, G. & Michalewicz, Z., "*A Genetic Algorithm for the Linear Transportation Problem*", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.21, No.2.
- [64] Weiss, S.M. & Kulikowski, C.A., *Computer Systems That Learn*, Morgan Kaufmann, 1990.
- [65] Weiss, S.M. & Kapouleas, I., "*An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods*", *Proceedings of the International Joint Conference on Artificial Intelligence*, 1989.
- [66] Wilson, S., "*Classifier Systems and the Animat Problem*", *Machine Learning*, 3:2, 1987.
- [67] Winston, P.H., *Artificial Intelligence*, Addison Wesley, 1984.
- [68] Wisniewski, E.J. & Anderson, J.A., "*Some Interesting Properties of a Connectionist Inductive Learning System*", *Proceedings of the 5th International Conference on Machine Learning*, 1988.
- [69] Wnek, J., Sarma, J., Wahab, A., Michalski, R.S., "*Comparing Learning Paradigms via Diagramatic Visualization*", *Methodologies for Intelligent Systems 5*, M. Emrich, Z. Ras, M. Zemankowa (eds), 1990.
- [70] Zhang, J. & Michalski, R.S., "*Rule Optimization via SG-TRUNC Method*", *Technical Reports*, AI Laboratory, GMU, 1989.