INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

INCORPORATING THE SIMPLICITY FIRST METHODOLOGY INTO A MACHINE LEARNING GENETIC ALGORITHM

Steven M. Winikoff

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montreal, Quebec, Canada

January 1999

Abstract

Incorporating the Simplicity First Methodology into a Machine Learning Genetic Algorithm

Steven M. Winikoff

The classical genetic algorithm provides a powerful yet domain-independent tool for concept learning. However, in general, learning systems based on the genetic algorithm generally do not perform as well as symbolic learning algorithms.

Robert Holte's symbolic learning algorithm 1R demonstrated that simple rules can perform well in non-trivial learning problems, and inspired an approach to machine learning which Holte termed "simplicity first research methodology".

A system called ELGAR is proposed, constructed and evaluated in order to investigate the properties of concept learning using the genetic algorithm. A hybrid algorithm is then developed and implemented which integrates the genetic algorithm in ELGAR with the "simplicity first" approach, resulting in a concept learning system that outperforms both 1R and the purely genetic version of ELGAR.

Acknowledgments

There are many people without whom this work either would not have begun, or would not have been completed. In particular, I would like to thank: my parents and Laurel Ladd, for encouraging me to do this in the first place; my supervisor, Dr. Rajjan Shinghal, for helping me to do it; and my family, friends and colleagues, both for their moral support and for putting up with me while I did it.

The data sets used for experimentation were graciously provided by the Ludwig Breast Cancer Study Group and by the University of California at Irvine, home of the UCI Repository of Machine Learning Databases and Domain Theories. The UCI Repository's librarians, Patrick M. Murphy, David Aha and Christopher J. Merz, deserve special recognition for the service they have provided and continue to provide to all researchers in the field of machine learning.

Finally, I would like to thank Micheline Kamber and Jennifer Scott for their assistance in understanding the Ludwig Breast Cancer Study Group data and preparing it for use; their expertise in both computer science and medicine was extremely helpful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine Learning

Machine learning is the study of computer systems which can learn. Although learning is more commonly considered to be a characteristic only of intelligent life, in practice there is strong motivation to create systems which can monitor the results of their current activity, and use this information to improve their future performance. There are many potential uses for this type of ability, including basic control systems (*e.g.* an automatic transmission that can customize its shift patterns to the habits of a particular driver), user interface design (*e.g.* an electronic mail reader which can learn how to rank messages in order of relevance based on the user's prior reading habits), and knowledge acquisition [Giarratano94] (*i.e.* the ability to learn if-then rules, based on sources such as medical or legal case history files).

1.1.1 The Machine Learning Problem

Just as there are many types of learning in general, there are many different definitions of and approaches to the task of machine learning. The manner in which the learning problem is defined will naturally influence the manner in which it is solved; thus a multiplicity of different learning programs exist, specialized for different types of learning tasks. Section 1.1.2 presents a broad categorization of general approaches to machine learning, and Section 1.2 describes a number of specific machine learning implementations.

1.1.2 General Approaches to Machine Learning

Micheline Kamber [Kamber91, pp. 33–50] describes a taxonomy of machine learning techniques, of which the following are the major categories:

1. *rote learning*, in which no inferencing is performed, and all new information is directly usable by the system

2. *learning by deduction*, a process of specialization which essentially consists of combining existing facts and rules to generate new facts or rules

3. *learning by induction*, in which general rules or concepts are drawn from existing, more specific facts or rules

4. *learning by analogy*, where the learning system attempts to deal with new situations by recourse to existing knowledge of similar ones

5. *explanation-based learning*, in which **generalization**[1] is performed from a single **instance**, often with the aid of a teacher and substantial background knowledge of the domain being studied

6. *hybrid learning*, which combines two or more of the above techniques in one system

Much recent research in machine learning is concerned with **concept learning**, which in general concerns the last four of the above approaches. In addition to the discussion in Section 1.1.3, two good overviews of this topic may be found in [Thornton92] and [Shavlik90].

1.1.3 Concept Learning

The term **concept learning** refers to learning a **concept** from a set of **instances** whose classification with respect to the target concept are known *a priori*. This can range from concepts which are relatively simple, such as 'red', 'chair' or 'conditions under which it is safe to cross the street', to concepts which are quite complex indeed,

[1] Throughout this thesis, **bold type** is used to denote terms which are defined in the glossary (Appendix A).

such as 'conditions under which a patient recovering from surgery for breast cancer will not have the cancer recur'.

The goal of a system which performs concept learning is to create a representation of the newly learned concept which would be capable of classifying heretofore unseen instances, called **recall instances**, and correctly determining whether or not they belong to the actual concept which was to be learned. Each instance consists of a number of observations, typically referred to as **features** or **attributes**, which describe various characteristics of the object to which they refer.

1.1.3.1 A Concept Learning Example

As an example, consider Table 1, which presents data from [Clark90], listing observations regarding a number of animals, some of which are known to be lions. Given these eight **training instances**, a concept learner would attempt to formulate a representation of the concept LION.[2] Typically this would be expressed in the form of classification rules, using the attributes Furry?, Age and Size,[3] although other formulations are also used (see Section 1.2 for some examples).

Specimen Number	Furry?	Age	Size	Classification
1	yes	old	large	lion
2	yes	young	medium	lion
3	yes	young	large	lion
4	no	young	large	not lion
5	yes	old	small	not lion
6	yes	young	small	not lion
7	no	young	small	not lion
8	no	old	large	not lion

Table 1: Training instances for the concept LION

A major part of the concept learning problem is finding the best such formulation for any given concept. Even if the nature of the representation is specified (*e.g.* rules), there may still be many possible solutions (*e.g.* different sets of rules with potentially different classification behaviour).

[2]Throughout this thesis, concept names are printed in SMALL CAPS.
[3]Throughout this thesis, attribute names are printed in sans-serif type.

For example, one set of rules which could be obtained from the training set in Table 1 is

Rule Set 1.1

```
IF      (Furry = yes) AND (Size = medium) THEN     lion
ELSE IF (Furry = yes) AND (Size = large)  THEN     lion
ELSE                                           not lion
```

This rule set correctly classifies all of the eight training instances. However, so does the following:

Rule Set 1.2

```
IF                          (Size = medium) THEN     lion
ELSE IF (Furry = yes) AND (Size = large)  THEN     lion
ELSE                                           not lion
```

Rule Sets 1.1 and 1.2 behave identically with respect to the **training set**, but they differ in the predictions they make concerning unseen **recall instances**. For example, consider an animal which is old, not furry, and of medium size: according to Rule Set 1.1, this animal would not be considered a lion; according to Rule Set 1.2, it would be a lion. Of course, there is no guarantee that such an animal really is a lion; thus, until one is encountered, and its classification established by other means, it is not possible to determine even in principle which of these two rule sets is the more accurate representation of the concept LION.

It is particularly important to note that a rule set may fail to classify recall instances correctly even when it correctly classifies all of the training instances. Intuitively, this is likely to occur if the training set is insufficiently representative (whether by virtue of being too small, or because the instances it contains are fundamentally different from those observed in the real world), although the quality of the rule set which was induced is also a factor. As an example of the latter, consider the following rule set:

Rule Set 1.3

```
IF      (Furry = yes) AND (Age = old)   AND (Size = large)  THEN     lion
ELSE IF (Furry = yes) AND (Age = young) AND (Size = medium) THEN     lion
ELSE IF (Furry = yes) AND (Age = young) AND (Size = large)  THEN     lion
ELSE IF (Furry = no)  AND (Age = young) AND (Size = large)  THEN not lion
ELSE IF (Furry = yes) AND (Age = old)   AND (Size = small)  THEN not lion
ELSE IF (Furry = yes) AND (Age = young) AND (Size = small)  THEN not lion
ELSE IF (Furry = no)  AND (Age = young) AND (Size = small)  THEN not lion
ELSE IF (Furry = no)  AND (Age = old)   AND (Size = large)  THEN not lion
ELSE                                                             not lion
```

Although this rule set does indeed classify all of the training instances correctly, it is unlikely to be useful for classifying recall instances. This particular behaviour is an example of **overfitting**, *i.e.* relying on details which are specific to the training set and unlikely to be useful in general.

The lion example also illustrates that not all attributes present in the training set are necessarily useful. In this particular case, Rule Sets 1.1 and 1.2 make no use of the **Age** attribute. This example is small enough to make it intuitively clear that this is a reasonable omission, since no mammal yet discovered changes its species as it grows older. However, in learning domains with many attributes, it is often quite difficult to determine which attributes are relevant.

Furthermore, it may happen that some attributes or pairs of attributes are not independent of each other. Consider **Age** and **Size** in the lion example; granted that age is not a characteristic which is likely be helpful in determining an animal's species, but even if it were, age and size are generally positively correlated in most mammals.

As a result, attribute selection is an important facet of concept learning, both in designing the data collection in the first place, and in filtering out irrelevant or unhelpful attributes during the actual learning process. While progress has been reported in this area (*e.g.* [Vafaie92, Vafaie94, Holmes95]), as yet there is no known method which guarantees that only the optimal set of attributes will be selected.

1.1.3.2 Rule Set Structure

The language in which potential concept representations are expressed has a significant effect on the manner in which a given concept learning program will function. In particular, this will determine the set of concept representations which the program

is capable of forming (and, by extension, those of which it is incapable) [Mitchell80]. In fact, this is one form of **bias**, which in general is the term used to denote the tendency of a learning program to prefer certain hypotheses over others [Shavlik90, pp. 45–56].

One commonly used representation language is that of rules in **disjunctive normal form**, or **DNF** [Shavlik90, pp. 48–49]. In this form, the left-hand side (**antecedent**) of a single rule consists of a number of **conjuncts**, each of the form `(attribute = value)`, all joined by the logical **AND** ($\wedge$) operator; a rule set consists of some number of these rules, which themselves are all implicitly joined by the logical OR ($\vee$) operator. For example:

Rule Set 1.4

```
IF                    (Size = medium) THEN lion
IF (Furry = yes) AND (Size = large)  THEN lion
```

Rule Set 1.4 is equivalent to Rule Set 1.2, under the *closed world assumption* [Giarratano94, p. 168] which excludes anything not explicitly included. Specifically, any animal which satisfies either of the two rules is considered to be classified as a lion by the rule set as a whole (since the two rules are implicitly ORed together), while any animal which satisfies neither rule is implicitly classified as not a lion.

Henceforth, rules of this type will be displayed in logic form, as in the following example, which is equivalent to Rule Set 1.4:

Rule Set 1.5

```
(Size = medium) → lion
(Furry = yes) ∧ (Size = large) → lion
```

When the number of conjuncts in each rule is limited, as is often the case in concept representation (*i.e.* at most one conjunct per attribute), the resulting variation is termed **k-DNF**, where 'k' denotes that the number of conjuncts may not exceed a particular constant value [Shavlik90, pp. 48–49]. When the k-DNF format is used for concept learning, typically the value of k will be the number of attributes.

1.1.3.3 Concept Learning Evaluation

A commonly used measure of the quality of a rule set obtained via concept learning is its **error rate** with respect to recall instances [Thornton92, p. 10]. Typically

the total set of classified instances available to the experimenter will be partitioned into a **training set** used for learning, and a **recall set** used for evaluation. A rule set's error rate is then the percentage of recall instances which it classifies incorrectly. The complement of a rule set's error rate is its **classification accuracy**, *i.e.* the percentage of recall instances which it classifies correctly.

1.2 Specific Approaches to Concept Learning

Broadly speaking, concept learning can be attempted by empirical or analytical means. The following examples represent some of the more widely studied techniques. This section will present only a brief overview; details of selected algorithms will be provided in Chapter 3.

1.2.1 Empirical Methods

1.2.1.1 Decision Trees

According to [Shavlik90, p. 46],

> A decision tree is a tree in which the nodes contain tests (*e.g.* of patient attributes) and the leaves indicate to which class to assign (*e.g.* `healthy`[4] or `diseased`).

Decision trees are used for classification, by following a path from the root to a terminal node. Each complete path to a terminal node behaves like a single classification rule, allowing a correspondence to be drawn between decision trees and rule sets. For example, a decision tree corresponding to Rule Set 1.2 is shown in Figure 1.

1.2.1.1.1 C4

C4 [Quinlan86, Quinlan87, Quinlan90, Quinlan93, Quinlan96] is a system based in part upon ID3 [Quinlan79], which generates decision trees from a set of training instances.

Like the decision tree data structure itself, C4's algorithm for building trees is recursive. At any given node, C4 will first determine whether all training instances

[4]Throughout this thesis, attribute value names are printed in `typewriter-style type`.

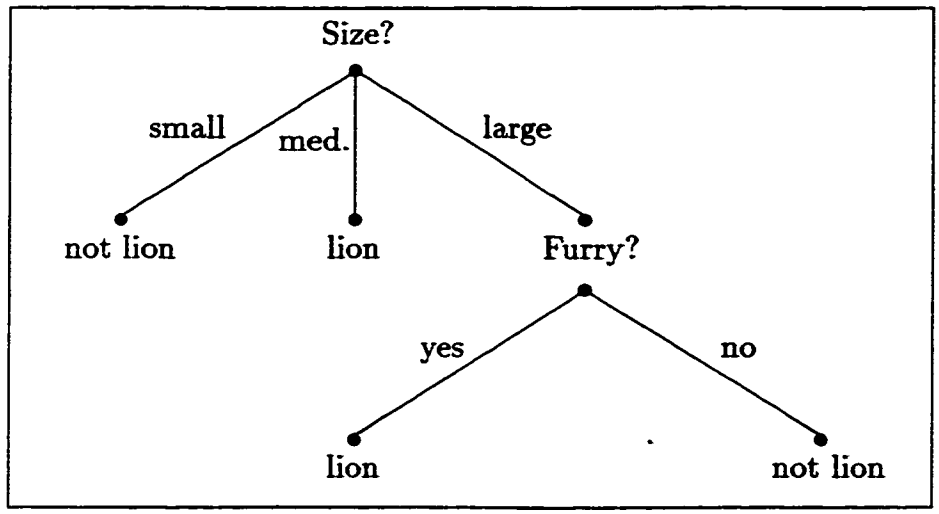

Figure 1: Decision tree corresponding to Rule Set 1.2

which the node describes share the same classification; if they do, that node will be a terminal node for that classification. Otherwise, an information theoretic measure which [Quinlan90] terms the *gain ratio criterion* will be used to determine which attribute to evaluate next. A more detailed description of this algorithm will be presented in Section 3.1. C4 generally performs very well, and in fact is widely regarded as a benchmark for concept learning systems.

1.2.1.1.2 IDA

Like C4, IDA [Tu92] is a decision tree generator based upon ID3 [Quinlan79]. IDA differs from ID3 and C4 in the criteria it uses to select the order in which attributes will be evaluated.

Attribute evaluation ordering is one of the principal problems in decision tree generation, since the same data set may lead to very different trees simply by examining the same attributes in a different order. For example, consider Figure 1, which evaluates the Size attribute first, followed, where necessary, by the Furry? attribute; had this tree evaluated Furry? first, the result would have corresponded to Rule Set 1.1 rather than Rule Set 1.2.

IDA's attribute ordering is based upon the *divergence measure*, for which Tu and Chung [Tu92] cite [Kullback59] and [Marill63], rather than C4's gain ratio.

Tu and Chung [Tu92] compare IDA to ID3 rather than to C4, and report that IDA is generally superior to ID3 for problems with small or medium sample sizes, and

for problems with dependent attributes (*e.g.* attributes such as **Age** and **Size** in the lion example, which may be correlated to some degree). When "data can be easily partitioned into more subgroups earlier" [Tu92], typically in large data sets, IDA tends to lose the advantage inherent in its use of the divergence measure, resulting in performance inferior to that of ID3.

1.2.1.2 Rule Induction

Rule induction systems are not characterized by a particular data structure as are decision tree generators; rather they tend to produce symbolic rules directly. Michalski's AQ family of algorithms are a typical example of this type of learner.

1.2.1.2.1 AQ15-TT

A distinguishing characteristic of AQ15-TT [Michalski90] is that it divides a concept description into two components: a *base concept representation*, or *BCR*, and an *inferential concept interpretation*, or *ICI*. Intuitively, the BCR is intended to capture the general case of the given concept, while the ICI describes exceptions or special cases. For this reason, the 'TT' portion of the algorithm's name is an acronym for 'two-tiered', while the 'AQ' portion derives from the fact that the BCR is built using AQ15, a recent member of a family of similar algorithms by Michalski [Michalski83].

The basic AQ algorithm attempts to learn a concept by explicitly creating rules that **cover** as many **positive instances** (*i.e.* instances which do belong to the given concept) as possible, without covering negative ones (*i.e.* those which do not belong to the concept), where an instance is said to be covered by a rule if it satisfies the conditions in the rule's antecedent. For each positive instance not already covered (which initially will be all positive instances available), the algorithm will select a single positive instance to start with (termed the *seed* by Michalski).

For each seed instance, rules are generated by starting with the most general rule set possible, namely '`TRUE` $\rightarrow$ `C`', where C is the concept to be learned. This rule set is then **specialized** so as not to cover any **negative instances.**

The specialization process operates on each existing rule in turn. For each attribute in an existing rule, a new rule is generated with a set of values for the given attribute that deliberately excludes each negative instance as it is encountered. The original rule is then replaced by the subset of newly generated rules which continues

to cover the seed instance. Specialization continues until as few negative instances as possible remain covered.

This straightforward approach works well in some domains, but in others it may happen that the specialization process is unable to progress far enough to produce a sufficiently accurate concept description; it is for this reason that Michalski developed the idea of the two-tiered representation, allowing AQ to describe the general case without needing to capture all possible exceptions.

1.2.1.3 Nearest Neighbour

The idea behind the *nearest neighbour* technique is that objects with similar classification will tend to be 'close' to each other in *instance space*. The precise definition of 'distance' in non-numerical domains is one of the key research questions in this area.

Given a method for measuring distance, a previously unseen instance can be classified by assuming the classification of whichever known instance is closest to it.

1.2.1.3.1 PEBLS

The PEBLS algorithm [Cost93] uses a distance measure based upon the *Value Difference Metric*, which Cost and Salzberg [Cost93] attribute to Stanfill and Waltz [Stanfill86]. However, their *Modified Value Difference Metric* or MVDM differs from the Stanfill-Waltz VDM in several minor ways, the most important of which being that it is symmetric. The MVDM distance between a pair of instances X and Y is defined as follows:

$$\Delta(X,Y) = w_X w_Y \sum_{j=1}^{m} \delta(x_j, y_j)^r$$

The terms w_X and w_Y are weights assigned to the two instances, m is the number of attributes recorded in each instance, and x_j and y_j are the values for attribute j in X and Y respectively; r is an arbitrary exponent, usually set to 2 [Cost93, p. 63]. Finally, $\delta(x_j, y_j)$ is the distance between the two specific values for attribute j, computed according to the formula:

$$\delta(V_1, V_2) = \sum_{i=1}^{n} \left| \frac{C_{1_i}}{C_1} - \frac{C_{2_i}}{C_2} \right|$$

Here, n is the number of classes (typically $n = 2$ for concept learning problems, where the two classes are **positive** and **negative**, *i.e.* the members and the non-members of the concept being learned). C_{1_i} is the number of instances classified as class i which contain value V_1 for the given attribute, and C_1 is the total number of instances in all classes which contain V_1; V_2, C_{2_i} and C_2 are analogous.

In other words, the distance between two instances is a weighted sum of the distance between their values for every attribute. The distance between two values for a given attribute is based upon the absolute difference between the relative frequencies of occurrence of the two values; this difference is computed for every possible class, and the results are added together to constitute the total distance.

Weights for stored instances (*i.e.* those already known to the system) are taken to be the ratio of the number of all uses of the given instance to the number of correct uses of that instance, where a stored instance is said to be used when it is found to be the closest possible match to the recall instance being evaluated. Stored instances which give rise to correct recall classifications will therefore have a weight approximating one, while those which do not will have weights with greater positive values. When a new training instance is stored, it is given the same weight as the neighbour (stored) instance which it matched.

In [Cost93], PEBLS is reported to be approximately as accurate as ID3 and *neural networks* (see Section 1.2.1.5) in most domains, and superior in some. It appears to be particularly robust when training set sizes are small, and has a tendency to handle exceptions well.

1.2.1.4 The Genetic Algorithm

The **genetic algorithm** [Goldberg89] was originally designed as a method for solving search and optimization problems, particularly those for which a mathematical model is unavailable or too difficult to solve. The approach was inspired by biological genetics, although it is not claimed to be an accurate model of any biological process.

In general terms, the algorithm begins with a set of guesses at the eventual solution. This set is typically referred to as the **population**, and may be initialized randomly if no better information is initially available.

The members of the population are evaluated according to some problem-specific criterion. The 'best' members are combined in some way to yield new members, which

then replace the original ones. The cycle of evaluation, combination and replacement continues until some arbitrary termination criterion is reached, at which time it will usually be the case that the final set of members is substantially 'better' than the original ones.

The genetic algorithm and its application to concept learning will be presented in greater detail in Chapter 4.

1.2.1.4.1 GABIL

Concept learning may be viewed as an optimization problem, namely that of finding the best possible concept description from among all those which can be expressed in a particular representation language.

GABIL [De Jong93] is a concept learning program based upon this idea, using a modified version of the genetic algorithm, which generally achieves results comparable to those of C4. Details of this algorithm will be presented in Section 4.4.2.

1.2.1.5 Neural Networks

According to [Caudill92, p. 3], a neural network

> consists of a number of very simple and highly interconnected processors called *neurodes*, which are the analogs of the biological neural cells, or neurons, in the brain. The neurodes are connected by a large number of weighted links, over which signals can pass. Each neurode typically receives many signals over its incoming connections; some of these incoming signals may arise from other neurodes, and others may come from the outside world — a photon striking a photoreceptor, for example, or an input signal pattern presented to the network by the designer. The neurode usually has many of these incoming signal connections; however, it never produces more than a single outgoing signal. That output signal transmits over the neurode's outgoing connection (corresponding to the biological axon of a neuron), which usually splits into a very large number of smaller connections, each of which terminates at a different destination. Each of these branches of the single outgoing connection transmits the same signal; the signal is not split or divided among them in any way.

> Most of these outgoing branches terminate at the incoming connection of some other neurode in the network; others may terminate outside the network and generate control or response patterns.

Essentially a neural network is a kind of classifier system, which maps an arbitrary pattern to a specific classification. If such a network is suitably organized, it can function in much the same way as a rule set or decision tree. In this case, one can think of the network as representing a learned concept, where the learning is expressed as the set of weights on the various connections between neurodes.

Neural networks are easy to use once designed; the training process can be very computationally expensive, and designing the correct network topology for a specific learning task is not by any means a solved problem. [Caudill92]

1.2.1.5.1 KBANN

One criticism often applied to neural networks is that, while they are capable of modelling concepts, the representation they use does not lend itself to easy interpretation; in particular, the network might pronounce that a given instance does or does not belong to a particular concept, but it would not be able to explain why or why not.

Towell and Shavlik [Towell90] have addressed this criticism by defining a class of neural networks which they refer to as *knowledge-based.* A knowledge-based neural network is one in which the network topology is based upon an existing set of rules; in particular, KBANN [Towell90] provides a method which allows any arbitrary rule set to be translated deterministically into an equivalent neural network [Towell90].

Once such a network has been created, it can be trained in the usual way. The resulting network, which in general will be a more successful classifier than it was before it was trained, may then be translated back into symbolic rules [Towell93]. This three-step process (translate, train, and translate back) allows existing rules to be improved by means of the network, without requiring the experimenter to define the network manually.

1.2.2 Analytical Methods

1.2.2.1 Explanation-Based Learning

In explanation-based learning, a single instance is presented in detail to the learning algorithm. This instance is then used by the system in order to generalize an existing domain theory, and the generalized version is abstracted in such a way as to allow it to be applied to new, *i.e.* recall, problems.

1.2.2.1.1 Physics 101

Physics 101 [Shavlik86] is a typical example of an explanation-based learner. In this particular case, the problem domain is Newtonian mechanics, and the system is provided with background knowledge, including a representation of Newton's laws, and the ability to perform "many of the mathematical manipulations expected of a college freshman" [Shavlik86, p. 307].

In operation, Physics 101 is presented with a problem. If it can solve the problem using its current knowledge base, it does so; otherwise, it asks the user for a solution. If a solution is given to it, it will verify that solution, requesting additional details if individual steps in the solution cannot be understood. Next it will explain the solution, generalize the result, and update its knowledge base accordingly.

1.3 Outline

The remainder of this thesis is organized as follows:

- Chapter 2 lists the data sets used for testing different concept learning systems, including their origin and a description of the attributes they contain.
- Chapter 3 discusses some existing learning programs, including a detailed description of their algorithms and their results on the data sets presented in Chapter 2.
- Chapter 4 describes the genetic algorithm, discusses how it has been applied to machine learning both in general and in particular, and then presents the ELGAR genetic learning system, which was developed for this thesis.

- Chapter 5 describes the extension of ELGAR into a hybrid algorithm that incorporates the "simplicity first" concept proposed in [Holte93].
- Chapter 6 summarizes the results obtained with the genetic and hybrid versions of ELGAR, and proposes some potential directions for further research.
- Appendix A is a glossary of terms introduced in this document.
- Appendix B is a verbatim transcript of a single multi-pass run of Hybrid ELGAR.

Chapter 2

Data Sets

Of the five data sets used for experimentation, the first four described below come from the UCI Repository of Machine Learning Databases and Domain Theories [Merz98], which is a collection of databases for machine learning maintained by the University of California at Irvine.

The fifth data set was provided by the Ludwig Breast Cancer Study Group, an international organization headquartered in Bern, Switzerland [Group89, Cavalli88]. This particular data set contains the results obtained in a clinical trial performed between November 1981 and December 1985. For reasons of medical confidentiality, the Ludwig data are available only with the permission of the Ludwig Breast Cancer Study Group, and must be obtained from them directly.

Two of the five data sets contain artificial data, while the remaining three are all drawn from real-world medical data dealing with breast cancer.

2.1 Mushroom Data

This data set was donated to the UCI Repository by Jeffrey Schlimmer [Schlimmer87], and is described as follows in [Merz97]:

> This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that

there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

For purposes of this work, instances of edible mushrooms were considered positive, while those of inedible mushrooms were considered to be negative.

The data set contains a total of 8,124 instances, of which every second instance was used for recall, and the remaining ones used for training. This results in a training set containing 2,125 positive and 1,937 negative instances; the recall set contains 2,083 positive and 1,979 negative instances.

In addition to its classification, each instance includes the 22 attributes shown in Table 2, all of which are **discrete**. A total of 2,480 instances are missing values for the stalk-root attribute.

The **baseline accuracy** of a training or recall set is the classification accuracy which one would expect to obtain just by guessing. In particular, this is the accuracy achieved by guessing that any arbitrary instance in the set will belong to whichever class (*i.e.* positive or negative) is more common in the set, and is computed by dividing the total number of instances in the set by the number of instances which actually do belong to the more common class. This figure represents the minimum acceptable classification accuracy for any learning algorithm.

In the case of the Mushrooms data set, the training and recall sets both contain a majority of positive instances. The training set contains 4,062 instances, of which 2,125 are positive and 1,937 are negative; since there are more positive instances than negative ones, the resulting baseline accuracy is

$$2,125 \div 4,062 = 0.523, \ i.e.\ 52.3\%$$

Similarly, the baseline accuracy for the recall set is

$$2,083 \div 4,062 = 0.513, \ i.e.\ 51.3\%$$

2.2 Data for the MONK's Problems

The MONK's Problems are a set of three artificial problem domains, which were created specifically for the purpose of comparing a number of different machine learning algorithms [Thrun91].

Attribute	Possible Values of the Attribute
cap-shape	bell, conical, convex, flat, knobbed, sunken
cap-surface	fibrous, grooves, scaly, smooth
cap-color	brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow
bruises?	bruises, no
odor	almond, anise, creosote, fishy, foul, musty, none, pungent, spicy
gill-attachment	attached, descending, free, notched
gill-spacing	close, crowded, distant
gill-size	broad, narrow
gill-color	black, brown, buff, chocolate, gray, green, orange, pink, purple, red, white, yellow
stalk-shape	enlarging, tapering
stalk-root	bulbous, club, cup, equal, rhizomorphs, rooted, missing
stalk-surface-above-ring	fibrous, scaly, silky, smooth
stalk-surface-below-ring	fibrous, scaly, silky, smooth
stalk-color-above-ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow
stalk-color-below-ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow
veil-type	partial, universal
veil-color	brown, orange, white, yellow
ring-number	none, one, two
ring-type	cobwebby, evanescent, flaring, large, none, pendant, sheathing, zone
spore-print-color	black, brown, buff, chocolate, green, orange, purple, white, yellow
population	abundant, clustered, numerous, scattered, several, solitary
habitat	grasses, leaves, meadows, paths, urban, waste, woods

Table 2: Attributes in the Mushrooms data set

All three problems are defined in terms of the same six discrete attributes, as listed in Table 3. No values are missing, but in problem M_3, described in Section 2.2.3, 5% of all training instances are deliberately given the wrong classification value, in order to test how various algorithms perform when confronted with **noise** in the training set [Merz93].

Attribute	Possible Values of the Attribute
a1	1, 2, 3
a2	1, 2, 3
a3	1, 2
a4	1, 2, 3
a5	1, 2, 3, 4
a6	1, 2

Table 3: Attributes in the MONK's Problems data set

The attributes used in the MONK's Problems support a total of 432 instances (obtained by multiplying together the number of possible values for each attribute, *i.e.* $3 \times 3 \times 2 \times 3 \times 4 \times 2$). Each individual problem uses its own set of randomly chosen instances for training, and uses all 432 instances for recall. These three training sets are not disjoint; each was created independently by selecting instances at random from the full collection of 432 possible instances, and thus some elements are common to all three sets.

2.2.1 Problem M_1

The concept to be learned in the first MONK's Problem [Merz93], expressed by Rule Set 2.1, is intended to be easily learned by symbolic learning algorithms [Thrun91, p. 2].

Rule Set 2.1

```
(a1 = a2) → member
(a5 = 1) → member
```

Note that this rule set requires a representation language capable of dealing with comparisons between attribute values. If the language in use is limited to comparisons of attribute values with literal values, then these rules must be rewritten as shown in Rule Set 2.2.

Rule Set 2.2

(a1 = 1) ∧ (a2 = 1) → member
(a1 = 2) ∧ (a2 = 2) → member
(a1 = 3) ∧ (a2 = 3) → member
(a5 = 1) → member

The training set for this problem contains 124 instances, chosen at random by Thrun *et al* [Thrun91, p. 2] from the universe of 432 possible instances. All 432 instances are used for recall. The baseline accuracy is exactly 50% for both the training and recall sets.

2.2.2 Problem M_2

The target concept for the second MONK's Problem can be stated as "exactly two of the six attributes have their *first* value" [Thrun91, p. 2].

This concept is deliberately designed to be difficult to learn by algorithms which use disjunctive normal form as a representation language. That this intention succeeds is demonstrated by Rule Set 2.3, which in fact expresses this concept in DNF.

Rule Set 2.3

(a1 = 1) ∧ (a2 = 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 = 1) ∧ (a2 ≠ 1) ∧ (a3 = 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 = 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 = 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 = 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 = 1) ∧ (a6 ≠ 1) → member
(a1 = 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 = 1) → member
(a1 ≠ 1) ∧ (a2 = 1) ∧ (a3 = 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 = 1) ∧ (a3 ≠ 1) ∧ (a4 = 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 = 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 = 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 = 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 = 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 = 1) ∧ (a4 = 1) ∧ (a5 ≠ 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 = 1) ∧ (a4 ≠ 1) ∧ (a5 = 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 = 1) ∧ (a4 ≠ 1) ∧ (a5 ≠ 1) ∧ (a6 = 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 = 1) ∧ (a5 = 1) ∧ (a6 ≠ 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 = 1) ∧ (a5 ≠ 1) ∧ (a6 = 1) → member
(a1 ≠ 1) ∧ (a2 ≠ 1) ∧ (a3 ≠ 1) ∧ (a4 ≠ 1) ∧ (a5 = 1) ∧ (a6 = 1) → member

The training set for this problem contains 169 instances, chosen at random by Thrun *et al* [Thrun91, p. 2] from the universe of 432 possible instances. All 432 instances are used for recall. The baseline accuracy is 62.1% (105 negative instances) for the training set, and 67.1% (290 negative instances) for the recall set.

2.2.3 Problem M_3

The target concept for the third MONK's Problem [Merz93], expressed by Rule Set 2.4, is not particularly difficult to learn. However, 5% of the instances in the training set are deliberately misclassified [Thrun91, p. 2].

Rule Set 2.4

(a4 = 1) ∧ (a5 = 3) → **member**

(a2 ≠ 3) ∧ (a5 ≠ 4) → **member**

The training set for this problem contains 122 instances, chosen at random by Thrun *et al* [Thrun91, p. 2] from the universe of 432 possible instances. All 432 instances are used for recall. The baseline accuracy is 50.8% (62 negative instances) for the training set, and 52.8% (228 **positive** instances) for the recall set.

2.3 Ljubljana Breast Cancer Data

This data set was provided by Dr. Matjaz Zwitter and Dr. Milan Soklic of the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. It was added to the UCI Repository in 1988 by Ming Tan and Jeffrey Schlimmer [Merz95].

This particular data set has been studied by many researchers, including Michalski [Michalski86], De Jong *et al* [De Jong93] and Holte [Holte93], among others.

Apart from the classification attribute, each instance contains the nine attributes listed in Table 4. Although [Merz95] reports that some attributes are **continuous**, in fact the data file available from the UCI Repository contains only **discrete** attributes. There are eight instances in which the value for attribute **node-caps** is missing, and one instance in which the value for attribute **breast-quad** is missing.

The data set contains a total of 286 instances, of which 201 are positive (classified as `no-recurrence-events`, *i.e.* that cancer did not recur for this patient), and 85 are negative (classified as **recurrence-events**).

As distributed, the data set is sorted such that all positive events are listed consecutively, followed by all negative events. For testing, instances were sorted randomly, then split such that every third instance was included in the recall set, with the remaining instances used for training.

Attribute	Possible Values of the Attribute
age	10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99
menopause	lt40, ge40, premeno
tumor-size	0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59
inv-nodes	0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39
node-caps	yes, no
deg-malig	1, 2, 3
breast	left, right
breast-quad	left-up, left-low, right-up, right-low, central
irradiat	yes, no

Table 4: Attributes in the Ljubljana Breast Cancer data set

The resulting training set contains 131 positive and 60 negative instances, for a baseline accuracy of 68.6%. The recall set contains 70 positive and 25 negative instances, for a baseline accuracy of 73.7%.

2.4 Wisconsin Breast Cancer Data

This data set was provided by Dr. William H. Wolberg of the University of Wisconsin Hospitals, Madison [Wolberg90]. It was donated to the UCI Repository in 1992 by Olvi Mangasarian [Merz92].

Apart from the classification attribute, and an identification number which is ignored for learning purposes, each instance contains the nine attributes listed in Table 5. There are 16 instances in which the value for attribute **Bare Nuclei** is missing.

Attribute	Possible Values of the Attribute
Clump Thickness	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Uniformity of Cell Size	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Uniformity of Cell Shape	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Marginal Adhesion	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Single Epithelial Cell Size	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bare Nuclei	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bland Chromatin	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Normal Nucleoli	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Mitoses	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Table 5: Attributes in the Wisconsin Breast Cancer data set

This data set has grown over time, as new clinical cases have been reported and added as new instances. The version used for this thesis contains a total of 699 instances, and has not been updated since 1992.

Instances are classified as `benign` (coded by the value 2, and interpreted as positive) or `malignant` (coded by the value 4, and interpreted as negative).

The training set contains 227 positive and 123 negative instances, for a baseline accuracy of 64.9%. The recall set contains 231 positive and 118 negative instances, for a baseline accuracy of 66.2%.

2.5 Ludwig Breast Cancer Study, Trial V

Trial V is the fifth in a series of clinical trials performed by the Ludwig Breast Cancer Study Group, an international organization headquartered in Bern, Switzerland. This particular trial occurred between November 1981 and December 1985, with a goal of studying the effect of "perioperative and conventionally timed chemotherapy in operable breast cancer" [Cavalli88, p. 466]. Specifically, the study's aim was to determine whether the time at which chemotherapy is applied is significant in averting relapses of the cancer.

The version of the data set supplied by the Ludwig group was last updated in September 1993, and includes followup observations of all patients up to that point. Four types of attributes are recorded:

1. *Administrative* attributes are not used in learning; these are items such as the patient identification number and date of entry into the trial. Quite apart from the fact that these attributes are easily recognized as being irrelevant, their use would cause overfitting to occur.

2. *Output* attributes record the outcome of each case, and as such are useful for formulating potential concepts to be learned. However, they are not suitable for use in training, since they include the specific information to be learned (*e.g.* the date when the cancer first recurred, which should not be made available to a program attempting to learn whether the cancer will in fact recur).

3. *Derived* attributes have values which are computed from the value of one or more other attributes. Attributes of this kind are often redundant and unlikely

to be useful for learning (*e.g.* the original data set includes both the measured size of the excised tumour, and the physician's classification of its size; either would be useful by itself, but having both provides little, if any, additional information).

4. *Input* attributes are those which do not fall into any of the above three categories. In particular, they represent information available to the physician at the time a patient undergoes treatment (including surgery and other therapy as deemed necessary). Examples of attributes of this kind include the patient's age, tumour size, or the elapsed time between surgery and the first application of chemotherapy.

The goal in learning from this data set is to be able to predict the outcome of a previously unseen case, based only upon the input attributes collected for that case, with respect to one or more output attributes. For the work reported in this thesis, the particular output attribute which was used is simply whether or not the cancer recurred during the followup period. Instances in which a relapse occurred were deemed to be positive, and those in which it did not were deemed to be negative.

The data set contains 2,504 instances, each consisting of the 44 attributes **f01** to **f44**, as shown in Table 6. In the table, the term **continuous** is used to denote an attribute with a numeric value that has no specific bound in at least one direction (*i.e.* an attribute with a value that may be unboundedly large or unboundedly small, or perhaps both). For some learning algorithms, including GABIL (see Section 4.4.2) and ELGAR (see Section 4.4.3), such attributes must be converted to a set of discrete values.

The following attributes in Trial V are derived from others [Kamber94]:

- Attribute **f15** is derived from **f14** according to the rule

$$f15 = \begin{cases} 0, & \text{if } f14 = 0 \\ 1, & \text{if } f14 > 0 \end{cases}$$

- Attribute **f18** is derived from **f19** according to the rule

$$f18 = \begin{cases} 0, & \text{if } 0 \leq f19 \leq 9 \\ 1, & \text{if } f19 \geq 10 \\ 2, & \text{if } f19 = -1 \end{cases}$$

- Attribute f20 is derived from f21 according to the rule

$$f20 = \begin{cases} 0, & \text{if } 0 \leq f21 \leq 9 \\ 1, & \text{if } f21 \geq 10 \\ 2, & \text{if } f21 = -1 \end{cases}$$

- Attribute f23 is derived from f22 according to the rule

$$f23 = \begin{cases} 1, & \text{if } f22 \leq 20 \\ 2, & \text{if } f22 > 20 \end{cases}$$

In order to prepare the data for use in machine learning, two corrections were applied to the file supplied by the Ludwig group:

- In the instance with f01 (case number) equal to 1,090, the value of f07 (overall survival time, measured in days) was less than the value of f09 (disease-free survival time, measured in days). Clearly this must have been a recording error, so the value of f07 was made equal to the value of f09.

- Some instances had negative values for f24 (the number of hours after mastectomy at which perioperative chemotherapy was applied). The specification for this attribute states that it must be a non-negative integer, with a value of 999 indicating cases in which perioperative chemotherapy was not applied. Thus, all such negative values were changed to 999.

Attribute f02 (patient group) is interpreted as shown in Table 7, encoding three different kinds of information simultaneously [Kamber94]: the patient's nodal status, menopausal status, and treatment type (where *CONCT* refers to conventionally timed chemotherapy, *PECT* refers to perioperative chemotherapy, *i.e.* chemotherapy applied immediately following surgery, *PECT+CONCT* refers to both types of chemotherapy, and *NO RX* refers to patients who did not receive chemotherapy at all). However, two of these items are already present in the data set, in other attributes (specifically nodal status, in f14, and menopausal status, in f16). Thus, repeating this information in f02 is redundant, which makes the learning task more difficult. As a result, a new attribute f45 was created to represent the treatment type afforded each patient, as shown in Table 8.

Number	Attribute	Type	Possible Values of the Attribute
f01	Case Number	admin	continuous
f02	Patient Group	input	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
f03	Month of Entry Into Study	admin	1, 2, 3, 4, 5, 6, 7, 8, 9, 10 11, 12
f04	Day of Entry Into Study	admin	1 - 31 inclusive
f05	Year of Entry Into Study	admin	81, 82, 83, 84, 85
f06	Clinic	input	14, 15, 16, 17, 20, 22, 23, 24, 25, 26, 27, 31, 32, 33, 34, 35, 36, 37, 38
f07	Overall Survival Time	output	continuous
f08	Alive or Dead	output	0, 1
f09	Disease-Free Survival Time	output	continuous
f10	Failure	output	0, 1
f11	Relapse-Free Survival	output	0, 1
f12	Site of First Failure	output	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
f13	Site of Worst Failure	output	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
f14	Nodes	input	continuous
f15	Nodal Status	derived	0, 1
f16	Menopause	input	0, 1, 2
f17	Age	input	continuous
f18	ER Status	derived	0, 1, 2
f19	ER Value	input	continuous
f20	PR Status	derived	0, 1, 2
f21	PR Value	input	continuous
f22	Tumour Size	input	continuous
f23	Tumour Size Classification	derived	1, 2
f24	MxPeri	input	continuous
f25	Days to Systemic Failure	output	continuous
f26	Systemic Failure	output	0, 1
f27	Leucovorin	input	0, 1
f28	Pathology Tumour Size	input	continuous
f29	Pathology Evaluated	input	0, 1
f30	Pathology Grade	input	0, 1, 2, 3
f31	Vessel Invasion	input	0, 1, 2, 3
f32	Nodes Sectioned	input	continuous
f33	Serial Section Done	input	0, 1
f34	Change to Node Positive	input	0, 1
f35	Multicentric	input	0, 1
f36	Histology	input	0, 1, 2, 3, 4, 5, 6, 7
f37	Invasive Ductal Subgroup	input	0, 1, 2, 3, 4, 5, 6, 7, 8
f38	Special Features Subgroup	input	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
f39	Medullary Tumour	input	0, 1
f40	CerbB2 Status	input	0, 1, 2
f41	H.Pomatia	input	0, 1, 2
f42	Amenorrhea Evaluability	input	0, 1, 2, 3, 4
f43	Amenorrhea cbt	input	0, 1, 2, 3
f44	Amenorrhea rec	input	0, 1, 2, 3, 9

Table 6: Attributes in the Ludwig Trial V data set

f02 Value	Nodes	Menopause	Treatment
1	negative	pre-menopausal	PECT
2	negative	pre-menopausal	NO RX
3	negative	post-menopausal	PECT
4	negative	post-menopausal	NO RX
5	positive	pre-menopausal	PECT
6	positive	pre-menopausal	PECT+CONCT
7	positive	pre-menopausal	CONCT
8	positive	post-menopausal	PECT
9	positive	post-menopausal	PECT+CONCT
10	positive	post-menopausal	CONCT

Table 7: Interpretation of the Ludwig Trial V 'Patient Group' attribute, f02

Treatment Type	(Input) f02 Values	(Output) f45 Value
PECT	1, 3, 5, 8	1
NO RX	2, 4	2
PECT+CONCT	6, 9	3
CONCT	7, 10	4

Table 8: Definition of the Ludwig Trial V 'Treatment Type' attribute, f45

Finally, the full data set was divided into two smaller subsets as described in Sections 2.5.1 and 2.5.2. The first of these is intended to provide a smaller version of the problem for experimentation, and contains attributes believed to be particularly relevant; the second subset all but the output and administrative attributes.

2.5.1 The 17-Attribute Subset

This version of Trial V includes 17 attributes, selected by Montreal medical expert Jennifer Scott, as shown in Table 9.

Instances were deemed positive if f11=1, and negative if f11=0; thus, the concept to be learned was that the cancer would eventually recur.

The 2,504 instances were divided such that every second instance was used for recall, with the remaining ones included in the training set.

The training set thus obtained contains 661 negative and 591 positive instances, for a baseline accuracy of 52.8%. The recall set contains 671 negative and 581 positive instances, for a baseline accuracy of 53.6%.

Number	Attribute	Type	Possible Values of the Attribute
f45	Treatment	derived	1, 2, 3, 4
f14	Nodes	input	0, 1-2, 3-4, >4
f16	Menopause	input	0, 1, 2
f17	Age	input	<31, 31-40, 41-45, 46-50, 51-55, 56-60, >60
f19	ER Value	input	-1, 0-9, >9
f21	PR Value	input	-1, 0-9, >9
f22	Tumour Size	input	<21, >20
f24	MxPeri	input	<13, 13-24, 25-36, 37-48, >48
f27	Leucovorin	input	0, 1
f30	Pathology Grade	input	0, 1, 2, 3
f31	Vessel Invasion	input	0, 1, 2, 3
f35	Multicentric	input	0, 1
f36	Histology	input	0, 1, 2, 3, 4, 5, 6, 7
f39	Medullary Tumour	input	0, 1
f40	CerbB2 Status	input	0, 1, 2
f43	Amenorrhea cbt	input	0, 1, 2, 3
f44	Amenorrhea rec	input	0, 1, 2, 3, 9

Table 9: Attributes in the Ludwig Trial V 17-attribute subset

2.5.2 The 31-Attribute Subset

This version includes all input and derived attributes except f23 (tumour size classification), and is intended to test the capacity of a learning algorithm to determine and use the most relevant attributes available, in the presence of a large pool of potentially irrelevant attributes. The specific set of attributes included is shown in Table 10.

The reason why f23 was excluded from this subset is that, after f22 (tumour size) is converted from continuous to discrete form, f22 and f23 are identical, at least theoretically. In practice, however, f22 is guaranteed to be converted correctly, since the conversion is done electronically, while f23 was created manually and therefore may contain transcription errors. In contrast, each of the other derived attributes is encoded in a manner which is different from the attribute upon which it is based, thereby retaining the potential to provide different information.

This data set includes the same set of instances in the training and recall sets as those used in the 17-attribute version (see Section 2.5.1), and therefore has the same baseline accuracy for each set.

Number	Attribute	Type	Possible Values of the Attribute
f02	Patient Group	input	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
f06	Clinic	input	14, 15, 16, 17, 20, 22, 23, 24, 25, 26, 27, 31, 32, 33, 34, 35, 36, 37, 38
f14	Nodes	input	0, 1-2, 3-4, >4
f15	Nodal Status	derived	0, 1
f16	Menopause	input	0, 1, 2
f17	Age	input	<31, 31-40, 41-45, 46-50, 51-55, 56-60, >60
f18	ER Status	derived	0, 1, 2
f19	ER Value	input	-1, 0-9, >9
f20	PR Status	derived	0, 1, 2
f21	PR Value	input	-1, 0-9, >9
f22	Tumour Size	input	<21, >20
f24	MxPeri	input	<13, 13-24, 25-36, 37-48, >48
f27	Leucovorin	input	0, 1
f28	Pathology Tumour Size	input	<21, >20
f29	Pathology Evaluated	input	0, 1
f30	Pathology Grade	input	0, 1, 2, 3
f31	Vessel Invasion	input	0, 1, 2, 3
f32	Nodes Sectioned	input	0, 1-5, 6-10, 11-15, 16-20, 21-25, 26-30, 31-35, 36-40, 41-45, >45
f33	Serial Section Done	input	0, 1
f34	Change to Node Positive	input	0, 1
f35	Multicentric	input	0, 1
f36	Histology	input	0, 1, 2, 3, 4, 5, 6, 7
f37	Invasive Ductal Subgroup	input	0, 1, 2, 3, 4, 5, 6, 7, 8
f38	Special Features Subgroup	input	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
f39	Medullary Tumour	input	0, 1
f40	CerbB2 Status	input	0, 1, 2
f41	H.Pomatia	input	0, 1, 2
f42	Amenorrhea Evaluability	input	0, 1, 2, 3, 4
f43	Amenorrhea cbt	input	0, 1, 2, 3
f44	Amenorrhea rec	input	0, 1, 2, 3, 9
f45	Treatment	derived	1, 2, 3, 4

Table 10: Attributes in the Ludwig Trial V 31-attribute subset

2.6 Side-by-Side Comparison

Table 11 compares the characteristics of all the data sets used, in their final form (in particular, after all continuous attributes have been converted to discrete values, where appropriate).

Data Set Name	Train Size	Recall Size	Train Baseline	Recall Baseline	# Attr.	# Attr. Values	# Missing
Mushrooms	4,062	4,062	52.3%	51.3%	22	125	2,480
MONK's 1	124	432	50.0%	50.0%	6	17	0
MONK's 2	169	432	(-) 62.1%	(-) 67.1%	6	17	0
MONK's 3	122	432	(-) 58.8%	52.8%	6	17	0
Ljubljana	191	95	68.6%	73.7%	9	51	9
Wisconsin	350	349	64.9%	66.2%	9	90	16
Trial V/17	1,251	1,251	(-) 52.8%	(-) 53.6%	17	63	880
Trial V/31	1,251	1,251	(-) 52.8%	(-) 53.6%	31	146	880

Table 11: Comparative overview of all data sets

The columns in Table 11 are interpreted as follows:

- The 'Train Size' and 'Recall Size' columns list the number of instances in the training and recall sets, respectively.
- The 'Train Baseline' and 'Recall Baseline' columns show the baseline accuracy of the training and recall sets, respectively. In all cases, the baseline outcome is positive unless indicated otherwise by the symbol '(-)' preceding the accuracy figure.
- The '# Attr.' column contains the number of attributes in this data set, as used for learning. Output and administrative attributes are not included in this value.
- The '# Attr. Values' column lists the total number of possible values for all attributes for the given data set, which is a finite number since all attributes are discrete. This value affects the run time of algorithms such as GABIL (Section 4.4.2) and ELGAR (Section 4.4.3), since it determines the number of bits needed to represent a single rule, and by extension the number of bits needed to encode a rule set.

- The '# Missing' column displays the total number of attribute values in the given data set which are missing or otherwise unknown. Different learning algorithms compensate for missing values in different ways; in general, however, the greater the number of missing values, the more difficult learning will be.

Chapter 3

Selected Symbolic Algorithms

3.1 C4

3.1.1 Background

C4 [Quinlan86, Quinlan87, Quinlan90, Quinlan93, Quinlan96] is a decision tree generator based in part upon ID3 [Quinlan79]. This particular program is one of the most widely respected and cited concept learning programs available, and is commonly used as a benchmark for new concept learning systems.

3.1.2 The C4 Algorithm

3.1.2.1 Outline

The main algorithm for C4, is shown in Figure 2. This material is drawn from [Quinlan90, p. 144] and [Quinlan93, pp. 17–18]. Quoted text in this figure is taken from [Quinlan90].

1. Build a decision tree:

   (a) Initialize the working set, which is a subset containing approximately 10% of all instances from the training set.

   (b) Create a tree based on the instances in the working set:

      i. Let k be the number of classes to be learned, denoted $C_1, C_2, ..., C_k$; let T be a set of training instances.

      ii. If T contains one or more instances which all belong to a single class C_j, then the decision tree is a leaf identifying class C_j.

      Else if T is an empty set, the decision tree is a leaf determined from information other than T (*e.g.* the class identified by the baseline rule for the data set, or the most frequent class at the parent of the current node).

      Else (T contains members of at least two different classes):

      A. Choose a test, based on a single attribute, that has one or more mutually exclusive outcomes $O_1, O_2, ..., O_n$; the attribute to use is the one with the maximum value for the gain ratio criterion.

      B. Partition T into subsets $T_1, T_2, ..., T_n$, where T_i contains all the instances in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and a single branch for each outcome O_i.

      C. Apply step (ii) recursively to each subset T_i, such that the i^{th} branch from T leads to the subtree constructed from T_i.

   (c) Test the tree using all training instances which are not members of the working set.

   (d) If any instances are misclassified, add at least half of the misclassified instances to the working set.

   (e) Repeat steps (a) - (d) until no improvement is possible.

2. Prune (*i.e.* simplify) the tree:

   (a) For each subtree:

      i. Form a pessimistic estimate of the subtree's error rate on recall (*i.e.* previously unseen) instances.

      ii. Estimate the error rate which would occur if this subtree were replaced by a leaf node.

      iii. If the difference between the two estimates is sufficiently small, replace the subtree by a leaf node.

3. Repeat steps 1 - 2 "several times", and select the "most promising" pruned tree.

Figure 2: Top-level pseudocode for C4

3.1.2.2 The Gain Ratio Criterion

At any non-leaf node in a decision tree, C4 must select an attribute on which to base a test for that node (see step (1)(a)(ii)(A) in Figure 2). Unless otherwise specified by the user, C4 will make this choice by maximizing a statistic called the *gain ratio criterion*, which is defined as follows [Quinlan93, pp. 20–24], [Quinlan96, pp. 78–79]:

- Let T be a set of training instances.
- Let k be the number of distinct classes of which the elements in T are members, and let these classes be denoted $C_1, C_2, ..., C_k$.
- Let T_j be the subset of instances in T which are members of class j.
- For any arbitrary set S, let $|S|$ denote the cardinality of S.
- Let $p(T,j)$ be the proportion of instances in T which belong to class C_j, which expresses the probability that a randomly selected instance in T belongs to C_j. By definition,

$$p(T,j) = \frac{|T_j|}{|T|}$$

- The *entropy* of a set S, denoted $Info(S)$, is defined as

$$Info(S) = -\sum_{j=1}^{k} f(S,j)$$

where

$$f(S,j) = \begin{cases} p(S,j) \times \log_2(p(S,j)), & \text{if } p(S,j) > 0 \\ 0, & \text{if } p(S,j) = 0 \end{cases}$$

[Quinlan96] refers to this quantity as "the residual uncertainty about the class to which a case in S belongs" (note that Quinlan uses the term 'case' to denote an instance).

- The *information gain* due to a test involving an attribute A with n unique values is defined as

$$Gain(T,A) = Info(T) - \sum_{i=1}^{n} \frac{|T_{A_i}|}{|T|} \times Info(T_{A_i})$$

where T_{A_i} is the subset of instances in T which have value i for attribute A.

- The "potential information obtained by partitioning a set of cases" [Quinlan96] with respect to attribute A is denoted $Split(T, A)$ and defined as

$$Split(T, A) = -\sum_{i=1}^{n} \frac{|T_{A_i}|}{|T|} \times \log_2 \left(\frac{|T_{A_i}|}{|T|} \right)$$

- The gain ratio criterion for attribute A is then defined as

$$Gain\_Ratio(T, A) = \frac{Gain(T, A)}{Split(T, A)}$$

3.1.3 C4 and the Lion Example

The following is intended to illustrate how C4 constructs a decision tree, using the 'lion example' from Section 1.1.3.1. The training instances for this data set are shown in Table 1.

3.1.3.1 Cardinality and Proportions of the Training Set

The first step is to compute the cardinality and proportions for the training set as a whole:

$$\begin{aligned} |T| &= 8 \\ p(T, \text{lion}) &= 3/8 \\ p(T, \text{not lion}) &= 5/8 \end{aligned}$$

3.1.3.2 Cardinality and Proportions per Attribute

Next, similar computations are done for the portions of the training set corresponding to every possible value of every attribute.

Furry?:

$$
\begin{array}{llllll}
|T_{yes}| & = & 5 & & & \\
|T_{no}| & = & 3 & & & \\
p(T_{yes}, \text{lion}) & = & 3/5, & p(T_{yes}, \text{not lion}) & = & 2/5 \\
p(T_{no}, \text{lion}) & = & 0/3, & p(T_{no}, \text{not lion}) & = & 3/3
\end{array}
$$

Age:

$$
\begin{array}{llllll}
|T_{old}| & = & 3 & & & \\
|T_{young}| & = & 5 & & & \\
p(T_{old}, \text{lion}) & = & 1/3, & p(T_{old}, \text{not lion}) & = & 2/3 \\
p(T_{young}, \text{lion}) & = & 2/5, & p(T_{young}, \text{not lion}) & = & 3/5
\end{array}
$$

Size:

$$
\begin{array}{llllll}
|T_{small}| & = & 3 & & & \\
|T_{medium}| & = & 1 & & & \\
|T_{large}| & = & 4 & & & \\
p(T_{small}, \text{lion}) & = & 0/3, & p(T_{small}, \text{not lion}) & = & 3/3 \\
p(T_{medium}, \text{lion}) & = & 1/1, & p(T_{medium}, \text{not lion}) & = & 0/1 \\
p(T_{large}, \text{lion}) & = & 2/4, & p(T_{large}, \text{not lion}) & = & 2/4
\end{array}
$$

3.1.3.3 Entropy for the Entire Training Set

The entropy for the training set as a whole is

$$
\begin{aligned}
Info(T) &= f(T, \text{lion}) + f(T, \text{not lion}) \\
&= -(p(T, \text{lion}) \cdot \log_2 p(T, \text{lion}) + \\
&\quad p(T, \text{not lion}) \cdot \log_2 p(T, \text{not lion})) \\
&= -(3/8 \cdot log_2(3/8) + 5/8 \cdot log_2(5/8) \\
&= 0.95443
\end{aligned}
$$

3.1.3.4 Entropy per Attribute Value

The entropy for each attribute value is

$$\begin{aligned} Info(T_{yes}) &= f(T_{yes}, \text{lion}) + f(T_{yes}, \text{not lion}) \\ &= -(3/5 \cdot log_2(3/5) + 2/5 \cdot log_2(2/5)) \\ &= 0.97095 \end{aligned}$$

$$\begin{aligned} Info(T_{no}) &= f(T_{no}, \text{lion}) + f(T_{no}, \text{not lion}) \\ &= -(0 + 3/3 \cdot log_2(3/3)) \\ &= 0 \end{aligned}$$

$$\begin{aligned} Info(T_{old}) &= f(T_{old}, \text{lion}) + f(T_{old}, \text{not lion}) \\ &= -(1/3 \cdot log_2(1/3) + 2/3 \cdot log_2(2/3)) \\ &= 0.91830 \end{aligned}$$

$$\begin{aligned} Info(T_{young}) &= f(T_{young}, \text{lion}) + f(T_{young}, \text{not lion}) \\ &= -(2/5 \cdot log_2(2/5) + 3/5 \cdot log_2(3/5)) \\ &= 0.97095 \end{aligned}$$

$$\begin{aligned} Info(T_{small}) &= f(T_{small}, \text{lion}) + f(T_{small}, \text{not lion}) \\ &= -(0 + 3/3 \cdot log_2(3/3)) \\ &= 0 \end{aligned}$$

$$\begin{aligned} Info(T_{medium}) &= f(T_{medium}, \text{lion}) + f(T_{medium}, \text{not lion}) \\ &= -(1/1 \cdot log_2(1/1) + 0) \\ &= 0 \end{aligned}$$

$$
\begin{aligned}
Info(T_{large}) &= f(T_{large}, \text{lion}) + f(T_{large}, \text{not lion}) \\
&= -(2/4 \cdot log_2(2/4) + 2/4 \cdot log_2(2/4)) \\
&= 1
\end{aligned}
$$

3.1.3.5 Information Gain per Attribute

The information gain for each attribute is

$$
\begin{aligned}
Gain(T, Furry?) &= Info(T) - (\frac{|T_{yes}|}{|T|} \cdot Info(T_{yes}) + \frac{|T_{no}|}{|T|} \cdot Info(T_{no})) \\
&= 0.95443 - (5/8 \cdot 0.97095 + 3/8 \cdot 0) \\
&= 0.34759
\end{aligned}
$$

$$
\begin{aligned}
Gain(T, Age) &= Info(T) - (\frac{|T_{old}|}{|T|} \cdot Info(T_{old}) + \frac{|T_{young}|}{|T|} \cdot Info(T_{young})) \\
&= 0.95443 - (3/8 \cdot 0.91830 + 5/8 \cdot 0.97095) \\
&= 0.00322
\end{aligned}
$$

$$
\begin{aligned}
Gain(T, Size) &= Info(T) - (\frac{|T_{small}|}{|T|} \cdot Info(T_{small}) + \\
&\quad \frac{|T_{medium}|}{|T|} \cdot Info(T_{medium}) + \\
&\quad \frac{|T_{large}|}{|T|} \cdot Info(T_{large})) \\
&= 0.95443 - (3/8 \cdot 0 + 1/8 \cdot 0 + 4/8 \cdot 1) \\
&= 0.45443
\end{aligned}
$$

3.1.3.6 Split Information per Attribute

The split information for each attribute is

$$\begin{aligned} Split(T, Furry?) &= -(\frac{|T_{yes}|}{|T|} \cdot \log_2\left(\frac{|T_{yes}|}{|T|}\right) + \frac{|T_{no}|}{|T|} \cdot \log_2\left(\frac{|T_{no}|}{|T|}\right)) \\ &= -(5/8 \cdot \log_2(5/8) + 3/8 \cdot \log_2(3/8)) \\ &= 0.95443 \end{aligned}$$

$$\begin{aligned} Split(T, Age) &= -(\frac{|T_{old}|}{|T|} \cdot \log_2\left(\frac{|T_{old}|}{|T|}\right) + \frac{|T_{young}|}{|T|} \cdot \log_2\left(\frac{|T_{young}|}{|T|}\right)) \\ &= -(3/8 \cdot \log_2(3/8) + 5/8 \cdot \log_2(5/8)) \\ &= 0.95443 \end{aligned}$$

$$\begin{aligned} Split(T, Size) &= -(\frac{|T_{small}|}{|T|} \cdot \log_2\left(\frac{|T_{small}|}{|T|}\right) + \frac{|T_{medium}|}{|T|} \cdot \log_2\left(\frac{|T_{medium}|}{|T|}\right) \\ &\quad + \frac{|T_{large}|}{|T|} \cdot \log_2\left(\frac{|T_{large}|}{|T|}\right)) \\ &= -(3/8 \cdot \log_2(3/8) + 1/8 \cdot \log_2(1/8) + 4/8 \cdot \log_2(4/8)) \\ &= 1.40564 \end{aligned}$$

3.1.3.7 Gain Ratio per Attribute

Finally, the gain ratio for each attribute is

$$\begin{aligned} Gain\_Ratio(T, Furry?) &= \frac{Gain(T, Furry?)}{Split(T, Furry?)} \\ &= 0.34759/0.95443 \\ &= 0.36419 \end{aligned}$$

$$\begin{aligned} Gain\_Ratio(T, Age) &= \frac{Gain(T, Age)}{Split(T, Age)} \\ &= 0.00322/0.95443 \\ &= 0.00337 \end{aligned}$$

$$
\begin{aligned}
Gain\_Ratio(T, Size) &= \frac{Gain(T, Size)}{Split(T, Size)} \\
&= 0.45443/1.40564 \\
&= 0.32329
\end{aligned}
$$

The attribute with the largest gain ratio is **Furry?**, which means that **Furry?** would be selected for the root node in the tree. Once this is done, there are now two subtrees to construct: one for the case where **Furry?** has the value **no**, and one where it has the value **yes**. The former case involves training instances 4, 7 and 8, all of which are not lions; this causes that branch of the tree to contain a terminal node with the label `not lion`. The latter case involves training instances 1, 2, 3, 5 and 6, some of which are lions (1, 2 and 3) and some of which are not (5 and 6). The partial tree at this point is as shown in Figure 3. The tree building procedure would then be invoked recursively with training instances 1, 2, 3, 5 and 6, considering only the remaining attributes **Age** and **Size**.

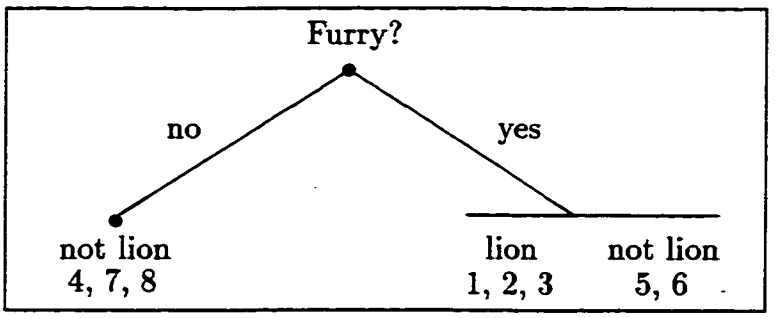

Figure 3: Partial decision tree produced by C4 for the lion example

Let T' denote the reduced training set containing instances 1, 2, 3, 5 and 6. Then the gain ratio computations in this phase are as follows:

$$
\begin{aligned}
|T'| &= 5 \\
p(T,' \text{lion}) &= 3/5 \\
p(T,' \text{not lion}) &= 2/5
\end{aligned}
$$

$$
\begin{aligned}
|T'_{old}| &= 2 \\
\left|T'_{young}\right| &= 3 \\
p(T'_{old}, \text{lion}) &= 1/2, \quad & p(T'_{old}, \text{not lion}) &= 1/2 \\
p(T'_{young}, \text{lion}) &= 2/3, \quad & p(T'_{young}, \text{not lion}) &= 1/3
\end{aligned}
$$

$$
\begin{aligned}
|T'_{small}| &= 2 \\
|T'_{medium}| &= 1 \\
\left|T'_{large}\right| &= 2 \\
p(T'_{small}, \text{lion}) &= 0/2, \quad & p(T'_{small}, \text{not lion}) &= 2/2 \\
p(T'_{medium}, \text{lion}) &= 1/1, \quad & p(T'_{medium}, \text{not lion}) &= 0/1 \\
p(T'_{large}, \text{lion}) &= 2/2, \quad & p(T'_{large}, \text{not lion}) &= 0/2
\end{aligned}
$$

$$
\begin{aligned}
Info(T') &= f(T', \text{lion}) + f(T', \text{not lion}) \\
&= -(3/5 \cdot log_2(3/5) + 2/5 \cdot log_2(2/5) \\
&= 0.97095
\end{aligned}
$$

$$
\begin{aligned}
Info(T'_{old}) &= f(T'_{old}, \text{lion}) + f(T'_{old}, \text{not lion}) \\
&= -(1/2 \cdot log_2(1/2) + 1/2 \cdot log_2(1/2)) \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
Info(T'_{young}) &= f(T'_{young}, \text{lion}) + f(T'_{young}, \text{not lion}) \\
&= -(2/3 \cdot log_2(2/3) + 1/3 \cdot log_2(1/3)) \\
&= 0.91830
\end{aligned}
$$

$$
\begin{aligned}
Info(T'_{small}) &= f(T'_{small}, \text{lion}) + f(T'_{small}, \text{not lion}) \\
&= -(0 + 2/2 \cdot log_2(2/2)) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
Info(T'_{medium}) &= f(T'_{medium}, \text{lion}) + f(T'_{medium}, \text{not lion}) \\
&= -(1/1 \cdot log_2(1/1) + 0) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
Info(T'_{large}) &= f(T'_{large}, \text{lion}) + f(T'_{large}, \text{not lion}) \\
&= -(2/2 \cdot log_2(2/2) + 0) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
Gain(T', Age) &= Info(T') - (\frac{|T'_{old}|}{|T'|} \cdot Info(T'_{old}) + \frac{|T'_{young}|}{|T'|} \cdot Info(T'_{young})) \\
&= 0.97095 - (2/5 \cdot 1 + 3/5 \cdot 0.91830) \\
&= 0.01997
\end{aligned}
$$

$$
\begin{aligned}
Gain(T', Size) &= Info(T') - (\frac{|T'_{small}|}{|T'|} \cdot Info(T'_{small}) + \\
&\quad \frac{|T'_{medium}|}{|T'|} \cdot Info(T'_{medium}) + \\
&\quad \frac{|T'_{large}|}{|T'|} \cdot Info(T'_{large})) \\
&= 0.97095 - (2/5 \cdot 0 + 1/5 \cdot 0 + 2/5 \cdot 0) \\
&= 0.97095
\end{aligned}
$$

$$
\begin{aligned}
Split(T', Age) &= -(\frac{|T'_{old}|}{|T'|} \cdot \log_2\left(\frac{|T'_{old}|}{|T'|}\right) + \frac{|T'_{young}|}{|T'|} \cdot \log_2\left(\frac{|T'_{young}|}{|T'|}\right)) \\
&= -(2/5 \cdot \log_2(2/5) + 3/5 \cdot \log_2(3/5)) \\
&= 0.97095
\end{aligned}
$$

$$
\begin{aligned}
Split(T', Size) &= -(\frac{|T'_{small}|}{|T'|} \cdot \log_2\left(\frac{|T'_{small}|}{|T'|}\right) + \\
&\quad \frac{|T'_{medium}|}{|T'|} \cdot \log_2\left(\frac{|T'_{medium}|}{|T'|}\right) + \\
&\quad \frac{|T'_{large}|}{|T'|} \cdot \log_2\left(\frac{|T'_{large}|}{|T'|}\right)) \\
&= -(2/5 \cdot \log_2(2/5) + 1/5 \cdot \log_2(1/5) + 2/5 \cdot \log_2(2/5)) \\
&= 1.52193
\end{aligned}
$$

$$
\begin{aligned}
Gain\_Ratio(T, Age) &= \frac{Gain(T, Age)}{Split(T, Age)} \\
&= 0.01997/0.97095 \\
&= 0.02057
\end{aligned}
$$

$$
\begin{aligned}
Gain\_Ratio(T, Size) &= \frac{Gain(T, Size)}{Split(T, Size)} \\
&= 0.97095/1.52193 \\
&= 0.63797
\end{aligned}
$$

Thus, the next attribute to be selected would be **Size**. The branch for `large` leads to a leaf node labelled `lion` (based on training instances 1 and 3), the branch for `medium` leads to a node labelled `lion` (based on training instance 2), and the branch for `small` leads to a node labelled `not lion` (based on training instances 5 and 6).

The resulting tree is as shown in Figure 4, and corresponds to Rule Set 1.1; it differs from the tree shown in Figure 1 in that the order of evaluation of attributes is inverted (although, like the tree in Figure 1, the **Age** attribute is not used).

3.2 1R

3.2.1 Background

1R is an atypical rule induction program. In particular, it was designed more to prove a point than to be the most accurate learning program possible. The chief

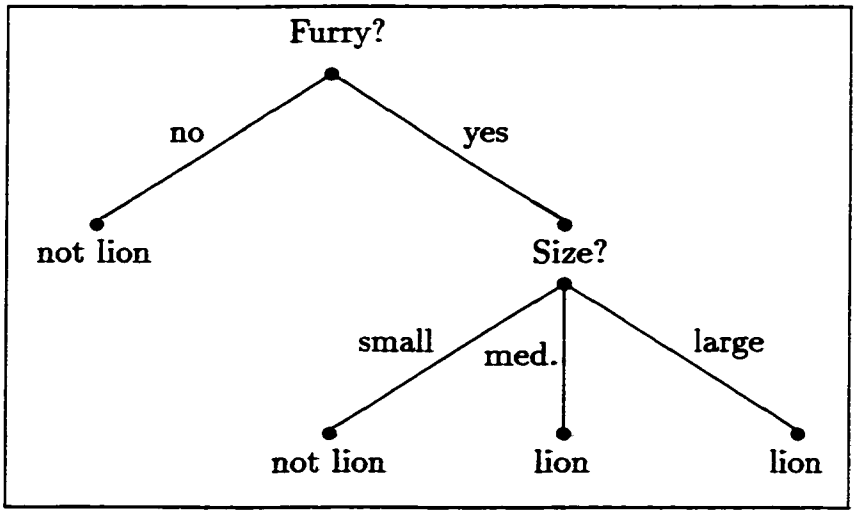

Figure 4: Decision tree produced by C4 for the lion example

characteristic of 1R is that, by design, it only creates **one-rules**, which are rules that evaluate one single attribute. Holte's [Holte93, p. 67] point in doing this was to demonstrate that such simple rules perform surprisingly well for many data sets which are commonly studied in machine learning, a fact which he attributes to the nature of the data sets involved.

3.2.2 The 1R Algorithm

The 1R algorithm as described in [Holte93, p. 77] is shown verbatim in Figure 5. Note that Holte uses the term "examples" to denote instances, and the term "nominal" to describe attributes with discrete values.

3.2.3 1R and the Lion Example

The following is intended to illustrate the working of the 1R algorithm, using the 'lion example' from Section 1.1.3.1. In this example, there are two classes to be learned (namely LION and NOT LION), three attributes which are already discrete, and a total of eight possible values.

3.2.3.1 Step 1: Build the COUNT Array

Given the instances shown in Table 1, the COUNT array acquires the values shown in Table 12. The 'Count' column in this table shows the number of instances which

1. In the training set, count the number of examples in class C having the value V for attribute A: store this information in a 3-D array, COUNT[C,V,A].

2. The default class is the one having the most examples in the training set. The accuracy of the default class is the number of training examples in the default class divided by the total number of training examples.

3. FOR EACH NUMERICAL ATTRIBUTE, A, create a nominal version of A by defining a finite number of intervals of values. These intervals become the "values" of the nominal version of A. For example, if A's numerical values are partitioned into three intervals, the nominal version of A will have three values "interval 1", "interval 2" and "interval 3". COUNT[C,V,A] reflects this transformation: COUNT[C,"interval I",A] is the sum of COUNT[C,V,A] for all V in interval I.

   Definitions:

   - Class C is optimal for attribute A, value V, if it maximizes COUNT[C,V,A].
   - Class C is optimal for attribute A, interval I, if it maximizes COUNT[C,"interval I",A].

   Values are partitioned into intervals so that every interval satisfies the following constraints:

   (a) there is at least one class that is "optimal" for more than SMALL of the values in the interval (this constraint does not apply to the rightmost interval); and

   (b) if V[I] is the smallest value for attribute A in the training set that is larger than the values in interval I, then there is no class C that is optimal both for V[I] and for interval I.

4. FOR EACH ATTRIBUTE, A, (use the nominal version of numerical attributes):

   (a) construct a hypothesis involving attribute A by selecting, for each value V of A (and also for "missing"), an optimal class for V (if several classes are optimal for V, choose among them randomly); and

   (b) add the constructed hypothesis to a set call HYPOTHESES, which will ultimately contain one hypothesis for each attribute

5. 1R: choose the rule from the set HYPOTHESES having the highest accuracy on the training set (if there are several "best" rules, choose among them at random).
   1R*: choose all the rules from HYPOTHESES having an accuracy on the training set greater than the accuracy of the default class.

Figure 5: The 1R Algorithm, reproduced from [Holte93, p. 77]

belong to the given class and have the given value for the given attribute; the numbers in parentheses following each count are the specimen numbers from Table 1 for those instances which contribute toward that count.

Class	Value	Attribute	Count
lion	yes	Furry?	3 (1, 2, 3)
lion	no	Furry?	0
lion	old	Age	1 (1)
lion	young	Age	2 (2, 3)
lion	small	Size	0
lion	medium	Size	1 (2)
lion	large	Size	2 (1, 3)
not lion	yes	Furry?	2 (5, 6)
not lion	no	Furry?	3 (4, 7, 8)
not lion	old	Age	2 (5, 8)
not lion	young	Age	3 (4, 6, 7)
not lion	small	Size	3 (5, 6, 7)
not lion	medium	Size	0
not lion	large	Size	2 (4, 8)

Table 12: Values of COUNT[C,V,A] for the lion example

3.2.3.2 Step 2: Find the Default Class

Of the eight training instances in Table 1, three are members of the class **lion**[1], and five are members of **not lion**. Thus, the default class is **not lion**, since it is the more common. The accuracy of the default class, which by definition is the baseline accuracy of the data set, is $5 \div 8 = 0.625$.

3.2.3.3 Step 3: Convert Continuous Attributes to Discrete Form

The 'lion example' has no continuous attributes, so this step does not apply.

3.2.3.4 Step 4: Construct Hypotheses

1R constructs one hypothesis for each attribute, basing each on the values in the COUNT array, as shown in Table 12. For each attribute, the COUNT entries are consulted in order to determine the class which is most common for each value of the

[1]Throughout this thesis, class names are printed in **sans-serif type**.

attribute, and this is the class assigned for that value in the hypothesis which will be built.

For example, consider the **Furry?** attribute, for which there are two values, **yes** and **no**. The relevant entries from the COUNT array are

```
COUNT[lion,     yes, Furry?] = 3
COUNT[not lion, yes, Furry?] = 2
COUNT[lion,     no,  Furry?] = 0
COUNT[not lion, no,  Furry?] = 3
```

For the value **yes**, instances which belong to the class **lion** outnumber those which belong to the class **not lion**; for the value **no**, the reverse is true. As a result, the hypothesis for **Furry?** is

```
(Furry = yes) → lion
(Furry = no)  → not lion
```

The **Age** and **Size** attributes are treated similarly, based on the values in Table 12. The resulting hypotheses for all three attributes are as shown in Table 13.

Value	Attribute	Hypothesis
yes	Furry?	(Furry = yes) → lion
no	Furry?	(Furry = no) → not lion
old	Age	(Age = old) → not lion
young	Age	(Age = young) → not lion
small	Size	(Size = small) → not lion
medium	Size	(Size = medium) → lion
large	Size	(Size = large) → lion (random)

Table 13: 1R hypotheses for the 'lion example'

Note that the training instances for which attribute **Size** has the value **large** are evenly split between the **lion** and **not lion** classes. Thus, 1R is unable to discriminate between the two classes for this particular value of this particular attribute, and so a class is selected at random. The **lion** class is the one shown in Table 13, but in practice it could just as well have been **not lion**.

3.2.3.5 Step 5: Choose a Hypothesis

The final output of the 1R algorithm will be a single hypothesis chosen from among those constructed in the previous step. Specifically, 1R will select the hypothesis with the greatest classification accuracy with respect to the training set.

The accuracy of each hypothesis is computed by counting the number of training instances which it classifies correctly, and dividing this result by the total number of training instances. For example, the hypothesis for the **Furry?** attribute classifies an instance as a **lion** if and only if it has the value **yes** for **Furry?**. According to Table 1, training instances 5 and 6 are not lions, but both have the value **yes** for the Furry? attribute; thus, these two instances are classified incorrectly by this hypothesis. The remaining six instances are classified correctly, leading to an accuracy figure of $6 \div 8 = 75\%$. Similarly, the classification accuracy of the hypotheses for the **Age** and Size attributes can be computed by inspecting the training instances and comparing their actual classification with that predicted by each hypothesis.

The resulting accuracy figures are shown in Table 14. The 'Accuracy' column lists the number of training instances classified correctly by each hypothesis, and the 'Training Instances' column shows the specimen numbers of the instances which are classified correctly.

Hypothesis	Accuracy	Training Instances
(Furry = yes) → lion (Furry = no) → not lion	6/8 = 75%	1, 2, 3, 4, 7, 8
(Age = old) → not lion (Age = young) → not lion	5/8 = 62.5%	4, 5, 6, 7, 8
(Size = small) → not lion (Size = medium) → lion (Size = large) → lion	6/8 = 75%	1, 2, 3, 5, 6, 7

Table 14: 1R hypothesis accuracy

The greatest observed accuracy is 75%, but there are two hypotheses with this accuracy figure. As a result, 1R will choose randomly between them. The final result of the 1R algorithm for the lion example would be Rule Set 3.1 or Rule Set 3.2, depending on the outcome of the random selection.

Rule Set 3.1

```
(Furry = yes)   → lion
(Furry = no)    → not lion
```

Rule Set 3.2

```
(Size = small)  → not lion
(Size = medium) → lion
(Size = large)  → lion
```

Note that 1R is less accurate than the rules in Rule Set 1.1; this is not unexpected, given that 1R makes no claim to produce the most accurate rules possible. However, 1R does guarantee that the rules it produces will be simple, and these rules tend to be almost as accurate as more complex rules derived by more intricate means.

3.2.4 "Simplicity First"

While one-rules will almost always not yield the best performance possible with respect to any given data set, the fact that they can come as close as they do led Holte to suggest a concept that he calls "simplicity first research methodology". If the machine learning problem is considered as a search through the space of all possible candidate rule sets (an idea which will be discussed in greater detail in Chapter 4), then typical concept learning programs attempt to search the entire space. Holte points out that this may not be necessary, and in fact may be counterproductive [Holte93, p. 75]:

> Complex hypotheses need not be considered for datasets in which most examples can be classified correctly on the basis of 1 or 2 attributes. An alternative, "simplicity first" methodology begins with the opposite premise: a learning system should search in a relatively small space containing only simple hypotheses. Because the space is small, navigating in it is not a major problem. In this methodology, progress occurs as researchers invent ways to expand the search space to include slightly more complex hypotheses that rectify specific deficiencies.

Essentially Holte is advocating a method that begins with the simplest useful approximation of a concept, and which adds complexity only when necessary to improve accuracy. This idea will be explored more fully in Chapter 5.

3.3 Results

Table 15 lists classification accuracy results obtained using C4 and 1R, with respect to the recall set for the data sets described in Chapter 2. All tests were performed using the training and recall sets described in Chapter 2.

Data Set	Baseline Accuracy	Recall Classification Accuracy	
		C4	1R
Ljubljana	73.7%	76.8%	72.6%
MONK's 1	50.0%	75.9%	75.0%
MONK's 2	67.1%	65.1%	67.1%
MONK's 3	52.8%	97.2%	80.6%
Mushrooms	51.3%	100.0%	98.4%
Trial V/17	53.6%	65.0%	64.3%
Trial V/31	53.6%	61.7%	64.3%
Wisconsin	66.2%	93.7%	92.3%

Table 15: C4 and 1R results

Note that the MONK's 2 and Ljubljana breast cancer data sets share the characteristic that the baseline accuracy of the training set is substantially lower than that of the recall set, as shown in Table 11; moreover, these are the only data sets studied for which this is true. This suggests that the training set for each problem is less representative of the recall set than is the case for the other six data sets, and in turn is most likely the reason why neither C4 nor 1R exceeds the baseline accuracy for MONK's 2, and why 1R does not reach the baseline accuracy for the Ljubljana breast cancer data.

3.4 Symbolic Learning Conclusions

From Table 15, we see that C4 clearly has difficulty with the deliberately awkward MONK's 1 and MONK's 2 data sets, but otherwise performs very well. The data sets studied here do, however, exhibit the properties described in [Holte93] which allow 1R to perform nearly as well as C4. In fact, 1R actually performs better than C4 in two of the eight data sets, and is within two percentage points of C4 on all but two of the remaining six data sets.

However, results for both programs for all three MONK's problems are disappointing compared to those published for other programs in [Thrun91], which suggests a motive for further exploration. Neither program does very well with the Ljubljana or Ludwig Trial V data sets, but this is likely to be due to the nature of the data rather than the programs in question.

Chapter 4

Genetic Learning

The term *genetic learning* is used to describe machine learning techniques based upon the genetic algorithm. The classical genetic algorithm was not designed for this particular use, having been developed as a method for solving search and optimization problems, particularly in **search spaces** for which a mathematical model is either unavailable or too difficult to solve analytically [Goldberg89, Beasley93, Whitley93].

However, concept learning can be recast in the form of a search problem (*e.g.* [De Jong93]), which permits the genetic algorithm to be used. Since the resulting search space is almost always both large and not well mapped (*i.e.* little is known about it *a priori*), it tends to be a good candidate for the genetic algorithm.

4.1 History of the Genetic Algorithm

Original development of the genetic algorithm is generally attributed to [Holland75], although Holland himself cites previous work in game theory going back as far as 1947. Holland began with the observation that many diverse fields depend on what he termed "adaptive processes", which occur both in 'natural' (*e.g.* biology and psychology) and 'artificial' fields of study (*e.g.* computational mathematics, economics and control problems). His goal was to "set up a mathematical framework which makes it possible to extract and generalize critical factors of the biological processes" [Holland75, p. v]. Holland's efforts culminated in a rigorous theoretical framework, which has been built upon by researchers such as De Jong (*e.g.* [De Jong75]) and Goldberg (*e.g.* [Goldberg89]), among others.

Although the classical form of the genetic algorithm was designed to solve search and optimization problems [Holland75], the scope of its application has broadened considerably over time. More recently, successful applications of the genetic algorithm to the problem of machine learning have been contributed by researchers such as Wilson [Wilson87], Bonelli *et al* [Bonelli90], McCallum and Spackman [McCallum90], Janikow [Janikow91], Michalewicz [Michalewicz92], Vafaie [Vafaie92, Vafaie94], De Jong *et al* [De Jong93] and Opitz and Shavlik [Opitz97].

Some current research topics involving the genetic algorithm itself, as opposed to applications or hybridizations of the basic technique, include:

- how best to represent and encode a given problem (*e.g.* [De Jong97, Spears98, Kazadi98, Tuson98, Patton98, Giguère98])
- finding isomorphisms or other relationships between the genetic algorithm and non-genetic approaches (*e.g.* [Vose98a, Vose98b])
- how to obtain the maximum benefit from parallel computing architectures (*e.g.* [Cantú-Paz97b, Cantú-Paz97a])

4.2 The Classical Genetic Algorithm

The genetic algorithm is not claimed to be an accurate model of any biological process. Nevertheless, its design was inspired by the process of biological evolution, in the sense that an initial population of candidate solutions to a problem can be improved by the simultaneous application of recombination techniques (*i.e.* the creation of a new candidate solution by combining information from two or more existing ones) and selection pressure.

4.2.1 Introduction

The classical algorithm is as shown in Figure 6. The end result of this process will be a final collection of candidate solutions, which in most cases will contain members that are significantly better than the ones generated initially.

1. Create an initial collection of candidate solutions. This is often, but not always, done randomly.
2. Evaluate each candidate.
3. Randomly select some number of the 'best' candidate solutions.
4. Combine the selected candidates in some way.
5. Repeat steps 2 - 4 until some arbitrary termination criterion is reached.

Figure 6: An informal description of the classical genetic algorithm

As an illustration, Figure 7 shows a hypothetical search space with an initial collection of randomly generated candidate solutions; Figure 8 shows how those candidates can converge toward the various optima in the search space (*i.e.* the various peaks in the curve), among them the global optimum. The curve shown in these figures does not represent any particular function; it is merely an arbitrary landscape in which the goal is to find the highest point. The difference between the two figures shows the evolution of the candidate solutions, which initially are placed at randomly chosen locations, but which ultimately tend to cluster near the individual peaks.

Figures 7 and 8 are an idealized representation of the process of **genetic search**, in which a large number of candidates are placed randomly into the search space, and then converge toward several local optima simultaneously; the point is to show how this process differs from traditional *hill climbing* algorithms (*e.g.* neural networks or simulated annealing [Caudill92]), which search only in the neighbourhood of a single point, and thus are more vulnerable to becoming trapped in the vicinity of a local optimum.

There is no guarantee that the optimal solution will be found (of course, it is important to realize that, in some problem domains, there may be no guarantee that an optimal solution even exists); however, in practice the algorithm will usually converge to a solution that is very nearly optimal. In particular, by maintaining a population rather than relying upon a single candidate solution at any given moment, the genetic algorithm is significantly less vulnerable to becoming lost in the vicinity of a local optimum than more traditional hill climbing methods such as neural networks [Goldberg89, Michalewicz92].

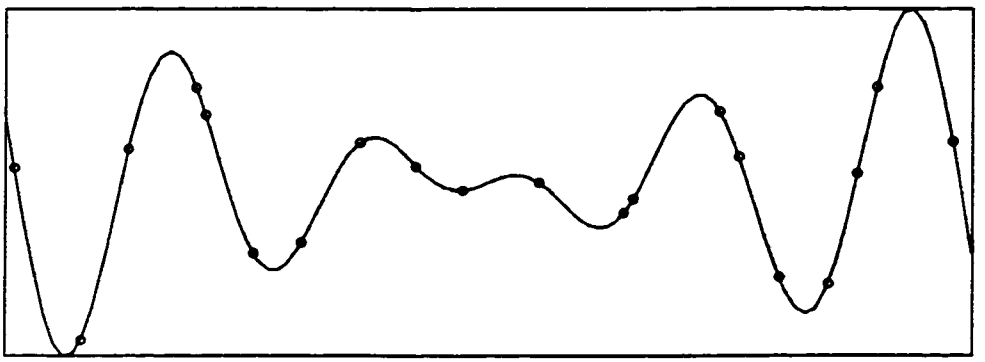

Figure 7: A sample search space, with some random guesses

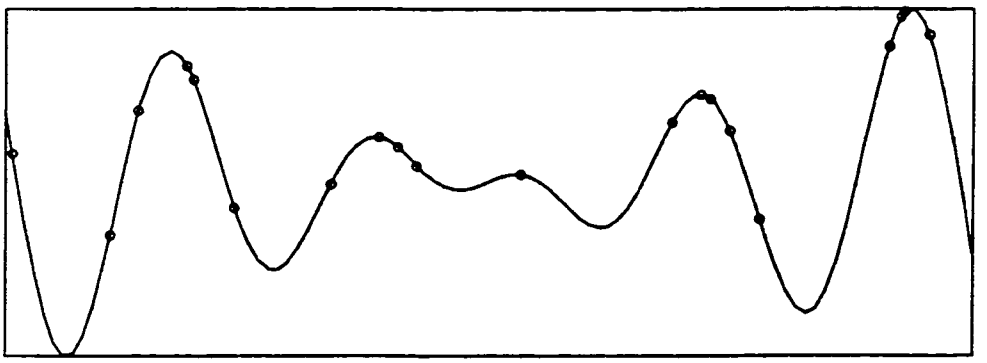

Figure 8: Sample search space with improved guesses

Before formalizing the above description, some terminology must be introduced:

- A single candidate solution is called an **individual.**
- A collection of individuals is called a **population.**
- A specific population at a given point in time is called a **generation.** Generations succeed each other as the individuals in the population at a particular time are replaced by new ones at the next cycle of the algorithm.
- The process of choosing individuals to combine is called **selection.**
- The combination process itself is called **reproduction.** This is composed of individual steps, using **operators** such as **mutation** and **crossover**, which are described in detail below.
- A **fitness function** is used to evaluate each individual as a potential solution. The resulting measurement is termed the individual's **fitness.**

Given this vocabulary, the classical genetic algorithm can be stated as shown in Figure 9 [De Jong93, p. 168]:

---

1. Let time t = 0.
2. Initialize population P(t).
3. Compute fitness for each individual in P(t).
4. Let t = t + 1.
5. Select new population P(t) from individuals in P(t-1).
6. Apply crossover operator to individuals in P(t).
7. Apply mutation operator to individuals in P(t).
8. Repeat steps 3 - 7 until some arbitrary termination criterion is reached.

Figure 9: The classical genetic algorithm

In practice the crossover and mutation operators are not applied deterministically, but instead each has a specific probability of application. This allows some individuals to pass unmodified from one generation to the next.

In order to apply the genetic algorithm to a particular search or optimization problem, the problem must first be described in a manner such that an individual will represent a potential solution, and a fitness function (*i.e.* a function which evaluates the quality of a given candidate solution) must be provided. These two areas are the only parts of the algorithm which are problem-specific [Goldberg89, Whitley93].

The classical genetic algorithm is designed to operate on individuals encoded as fixed-length bit strings, although more recent work (*e.g.* [Janikow91, Michalewicz92]) has been done using other encodings. Goldberg [Goldberg89, p. 15] provides an example in which the goal is to maximize the function $f(x) = x^2$ over the interval $0 \leq x \leq 31$. In this case, the solution will be an integer in the range from 0 to 31, giving rise to the natural encoding of allowing a five-digit binary string to represent its own integer value. In more challenging problem domains, the task of finding a suitable binary representation may not be straightforward; nevertheless the choice of representation will have a significant effect on the quality of the solutions which the genetic algorithm will be able to find [Goldberg89].

The fitness function's purpose is to influence the number of times a given individual will be selected for reproduction. As such, it must be based upon the individual's quality as a potential solution. In Goldberg's x^2 example, the obvious choice is the specific function being optimized, namely $f(x) = x^2$ itself. Of course, in the more general case, the genetic algorithm is needed most in those problem domains for which a mathematical function describing the domain does not exist. In cases of this kind, the user must find a function which will approximate the desired result as closely as possible, while simultaneously being as easy as possible to compute, since the function will be invoked a significant number of times.

4.2.2 Selection

The selection step builds a **mating pool** of individuals chosen from the current generation, as an intermediate step toward building the next generation. The individuals in this pool must be chosen randomly, but nevertheless the choice must be biased in favour of more highly fit individuals [Goldberg89].

The earliest genetic algorithms used *proportionate reproduction* [Goldberg91], in which an individual's probability of being selected is directly proportional to its fitness. Another common method is **binary tournament selection**, in which two

individuals are chosen at random, and the one with the greater fitness enters the mating pool. A variation on this method assigns a small probability that the individual with the lesser fitness will be selected instead.

Analysis of binary tournament selection finds that it performs similarly to other methods, but with lower computational complexity [Goldberg91, p. 90].

4.2.3 Mutation

The mutation operator inverts a randomly selected bit in an individual. For example, if an individual is represented by the bit string 10011, the mutation operator might invert the second bit from the right, changing the string to 10001.

The purpose of mutation is to introduce new strings into the population. However, mutation is usually employed with a very low probability; a typically used value is 0.001 (*e.g.* [Goldberg89, De Jong93]).

4.2.4 Crossover

The crossover operator requires two individuals as input (typically called **parents**), and produces two new individuals as output (typically called **offspring** or **children**). In the simplest version, a position within the bit string is selected at random, and all bits on one side of the chosen position are exchanged between the two individuals. For example, consider the following two parent individuals:

11111
00000

If bit position two (counting from the left, beginning at position zero) is selected as the crossover site, the result of this particular crossover would be

11100
00011

Variations on this operator include *two-point crossover*, in which two positions are selected and everything between them exchanged, and *uniform crossover*, in which every bit position in the string is exchanged with probability 0.5. Finally, Eshelman

introduced a variation on uniform crossover in which exactly half of the bit positions that differ in the two parents are exchanged between them [Eshelman91]; this variation was independently reported by Pawlowsky shortly thereafter [Pawlowsky92, Pawlowsky95]. The crossover operator is commonly applied with a probability of 0.6 (*e.g.* [Goldberg89, De Jong93]).

4.2.5 A Sample Run

The following is intended to illustrate the working of the classical genetic algorithm, given Goldberg's example of maximizing $f(x) = x^2$ over the interval $0 \leq x \leq 31$. The problem is encoded such that a five-digit binary string represents its own integer value, with the fitness function being $f(x) = x^2$.

Individual	Bits	Integer Value	Fitness
1	01001	9	81
2	01100	12	144
3	11010	26	676
4	01011	11	121
5	01111	15	225
6	00101	5	25

Table 16: Initial population for the x^2 example

Following the steps shown in Section 4.2.1, we begin by setting time t to zero. Next, we choose an initial population at random. For the sake of illustration, assume that the resulting population is as shown in Table 16. The fitness values shown in this table are computed by first converting the randomly initialized bit string to its own value as an integer, and then applying the fitness function x^2 to that integer.

We now increment t to one, and apply a selection method. Since our population size is six, we must choose six individuals to form the basis of the new generation. Using binary tournament selection as an example, we begin by generating twelve random numbers in the closed interval $[1, 6]$. For the sake of discussion, assume that the random sequence in question is $\{2, 3, 6, 2, 1, 4, 6, 1, 5, 3, 4, 5\}$. The six pairs resulting from this sequence, and the pool of selected individuals they generate, are shown in Table 17.

First Competitor		Second Competitor		Tournament
Number	Fitness	Number	Fitness	Winner
2	144	3	676	3
6	25	2	144	2
1	81	4	121	4
6	25	1	81	1
5	225	3	676	3
4	121	5	225	5

Table 17: Initial selection for the x^2 example

At this point the weakest individual, number six, has already been eliminated. For notational convenience, the individuals in the mating pool will now be renumbered as shown in Table 18.

Original Number	New Number	Bits	Integer Value	Fitness
3	1	11010	26	676
2	2	01100	12	144
4	3	01011	11	121
1	4	01001	9	81
3	5	11010	26	676
5	6	01111	15	225

Table 18: Mating pool at time $t = 1$ for the x^2 example

The next step is to apply the crossover operator, which works on a pair of individuals at a time. The pairs in question are chosen randomly from the mating pool, and are passed on unchanged with probability 0.4 (or, in other words, crossover is applied with probability 0.6). For the sake of discussion, assume that the random sequence used to select the next six individuals (*i.e.* three pairs of individuals) is $\{1, 3, 5, 6, 4, 4\}$. The one-point crossover operator will then be applied as shown in Table 19.

Pair	First Parent		Second Parent		Cross	Apply?	First	Second
Number	Number	Bits	Number	Bits	Site		Child	Child
1	1	11<u>010</u>	3	01<u>011</u>	1	yes	11<u>011</u>	01<u>010</u>
2	5	1101<u>0</u>	6	0111<u>1</u>	3	no	1101<u>0</u>	0111<u>1</u>
3	4	010<u>01</u>	4	010<u>01</u>	2	yes	010<u>01</u>	010<u>01</u>

Table 19: Crossover applied at time $t = 1$ for the x^2 example

Remarks concerning Table 19:

- The 'Cross Site' column shows the bit position immediately to the left of the point where the parent individuals will be cut, with bit positions numbered from the left starting at zero.
- The 'Apply?' column indicates whether the crossover operator is actually applied to the pair of individuals in question. This will depend on the operator's probability of application, and on a random number generated specifically to resolve this question. The values shown in the table are arbitrarily chosen for purposes of illustration.
- Note that the third pair involves the same parent individual twice. By definition any crossover operator applied to such a pair will be a null operation, resulting in the children being identical to their parents. This follows from the fact that crossover merely exchanges information between two individuals, but never creates any new information. In practice, allowing this situation to occur has the effect of slightly diminishing the operator's effective probability of application.

Once the crossover operator has been applied to all pairs of parents, the resulting offspring replace their parents in the mating pool. At this point the pool will be as shown in Table 20.

Number	Original Bits	New Bits	New Value	Fitness
1	11010	11011	27	729
2	01100	01100	12	144
3	01011	01010	10	100
4	01001	01001	9	81
5	11010	11010	26	676
6	01111	01111	15	225

Table 20: Mating pool after crossover at time $t = 1$ for the x^2 example

The mutation operator would next be applied to all parents in the mating pool. However, its typically low probability of application will usually result in no changes being made. Assuming that that would be the case in this situation, we end up with the mating pool as shown in Table 20 being the final contents of generation number one (where the initial population is denoted as generation number zero).

The algorithm will continue in this fashion for as many generations as possible until the termination criterion is reached. Typical criteria include some combination of the following:

- a predetermined number of generations
- a predetermined number of seconds of CPU time
- a lack of improvement over a specific number of consecutive generations (where improvement is typically measured as an increase in the average fitness over all members of the population)
- observation of an individual with a fitness value which exceeds a predetermined threshold

It is interesting to note that the single new generation created in this example resulted in a significant increase in average fitness. Specifically, the average fitness of the original population was

$$(81 + 144 + 676 + 121 + 225 + 25) \div 6 = 212$$

while the average fitness of generation one is

$$(729 + 144 + 100 + 81 + 676 + 225) \div 6 = 325.8\overline{33}$$

However, the best individual found in generation one (in particular, individual number one, with a bit pattern of 11011 and a corresponding value of 27) is only slightly better than the best individual in the initial population (individual number three, which has a bit pattern of 11010 and a value of 26).

As new generations are created, the population will tend to converge as more and more individuals in it tend to become copies or near copies of the most fit individual found thus far [Goldberg89]. If this phenomenon occurs too soon, the result is termed **premature convergence**, and it will typically be the case that the best solution found is only a local optimum. If premature convergence can be avoided, however, then the algorithm will usually converge to a solution which is close to the global optimum [Goldberg89]. In particular, with binary tournament selection, the expected number of generations until convergence occurs is proportional to $\log n$, where n is the population size [Goldberg94, p. 4].

Typical implementations of the genetic algorithm provide several operating parameters which can be adjusted, such as the population size, application probabilities for crossover and mutation, type of crossover to be used, selection method to be used, and the nature of the termination criterion. However, the primary influence on the success of the genetic algorithm in any particular problem domain is the correct choice of encoding and fitness function [Goldberg89].

4.2.6 Justification

It may appear counter-intuitive that a process which relies to so great an extent on randomness can actually achieve results in a reliable fashion. However, the genetic algorithm does indeed have a sound theoretical basis, established initially by [Holland75], and developed and explained in [Goldberg89, Beasley93, Whitley93] among other sources.

Informally, the power of the genetic algorithm derives from its use of both mutation and recombination [Goldberg98]; the former explores new areas of the search space, while the latter exploits knowledge already gained. This combination of exploration and exploitation is necessary in order for the search to be effective, particularly when the search space is large, multimodal and noisy [Goldberg89, Louis93].

Sections 4.2.6.1 and 4.2.6.2 present a summary of the groundwork elaborated in [Holland75] and [Goldberg89].

4.2.6.1 The Schema Theorem

In [Holland75], the term **schema** (plural 'schemata') is introduced to refer to a template that describes potential values for an individual. For binary-valued individuals, schemata are composed of the values 0, 1, and *, which denotes that a given bit position in the individual is free to assume either of the values 0 or 1 (informally referred to as a *don't care* symbol). For example, for individuals consisting of five-bit binary strings, the schema `1***0` would describe any individual which begins with a one and ends with a zero; similarly, the schema `01*1*` would describe only the individuals `01010`, `01011`, `01110` and `01111`.

Schemata have two properties of particular interest:

- The **order** of a schema S, denoted by $o(S)$, is the number of fixed positions in the schema (*i.e.* the number of positions not containing the * symbol).
- The **defining length** of schema S, denoted by $\delta(S)$, is the distance between the first and last fixed positions in S.

For example, if S is `0**1*`, then $o(S) = 2$ and $\delta(S) = 3$ (since the first fixed position is 0 and the last fixed position is 3). Similarly, if S is `**1**`, then $o(S) = 1$ and $\delta(S) = 0$.

Since schemata are defined over an alphabet containing three symbols, there exist 3^L possible schemata of length L. Moreover, any individual of length L is described by a total of 2^L of these schemata (since, for every possible bit position, a given schema may contain either * or the bit which actually occurs at that position in the given individual). It follows that a population of size n is described by a minimum of 2^L schemata (if all individuals in it are identical) and a maximum of $n \cdot 2^L$. Of course, some of these schema are 'better' than others, in the sense that they describe individuals which are more highly fit.

As the selection and recombination processes take place during the operation of the genetic algorithm, schemata in each generation are modified and, to a greater or lesser extent, replaced in the next generation. If the new schemata tend to be more fit than the ones they replace, then the average fitness of the population will increase over time.

In fact, this is exactly what happens. Schemata of higher fitness increase, and those of lower fitness decrease in quantity in succeeding generations; not only does

this process occur, it does so at an exponential rate. The cornerstone of genetic algorithm theory is a result which is formally stated and proven in [Holland75], and which Goldberg [Goldberg89, p. 33] paraphrases as follows:

> Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations. This conclusion is important, so important that we give it a special name: the *Schema Theorem*, or the Fundamental Theorem of Genetic Algorithms.

The consequence of the Schema Theorem is that, despite the randomness inherent in its implementation, the genetic algorithm does indeed proceed in a deterministic fashion from its initial state to one which is more highly fit. Moreover, the manner in which it does so takes advantage of the fact that many schemata are sampled simultaneously, an effect which [Holland75] terms "intrinsic parallelism".

4.2.6.2 The Building Block Hypothesis

Schemata which have a short defining length, low order and above-average fitness are termed **building blocks**. The *Building Block Hypothesis* suggests that "short, low-order, and highly fit schemata are sampled, recombined and resampled to form strings of potentially higher fitness" [Goldberg89, p.41].

This view of the actual mechanism by which improvement occurs across generations in the genetic algorithm provides insight into why not all potential encodings of a given problem will achieve equal results. In particular, every effort should be made to allow the optimal solution to be represented by schemata of short defining length, in order to allow the Building Block Hypothesis to apply to the maximum possible extent.

The importance of building blocks having a short defining length is that this makes them less vulnerable to being disrupted by the process of recombination. As the defining length of the schema which describes the optimal solution increases, it becomes less likely to be created by crossover or mutation, and more likely to be broken up if it does occur.

4.3 General Application of the Genetic Algorithm to Machine Learning

The key to applying the genetic algorithm to the problem of concept learning is to find a way to cast concept learning in the form of a search problem. In general terms, one way to do this is to consider concept learning as a search through the space of all possible concept descriptions for the one which most accurately classifies the available training instances.

As with any application of the genetic algorithm, the first step is to decide what an individual will represent, and how it will be encoded. Broadly speaking, if concepts are to be formulated as rule sets, then two representations are possible. The first is to consider each individual to be a single rule, in which case the entire population constitutes the rule set which will define the concept; the second is to consider each individual to be a complete rule set in itself. The former method has come to be known as "the Michigan approach", after work done by Holland and his students at the University of Michigan; the latter has come to be known as "the Pittsburgh approach", after work done by De Jong and his students at the University of Pittsburgh [De Jong90, pp. 625–628], [Michalewicz92, pp. 217–223]. Examples of rule sets in these two formats are presented in Sections 4.4.2.1.3 and 4.4.2.1.4, respectively.

The Michigan approach was motivated by an attempt to model human cognition [De Jong90, p. 627], and typically involves an elaborate model called a *classifier system* [Goldberg89], of which the genetic algorithm is only a part. The BOOLE and NEWBOOLE systems [Wilson87, Bonelli90] are well-known examples of this type. Although these and other systems have achieved some success [Goldberg89], De Jong argues that the classifier approach involves unnecessary complexity, and that as a result the Pittsburgh approach is "the more obvious and 'natural' way to proceed" [De Jong90, p. 625].

4.3.1 Individuals vs. Rules

When machine learning is performed using the genetic algorithm, the members of the population can be viewed on two distinct levels. From the point of view of the genetic search itself, each population member is nothing but a string of bits with arbitrary values; from the point of view of the learning process, each population member is either a single rule (in the Michigan format) or a complete rule set (in the Pittsburgh format).

The only component of the genetic algorithm that depends on the interpretation of each bit string is the fitness function, which in the case of machine learning must somehow evaluate each rule or rule set. It may also be possible to define additional genetic operators which would be specific to the task of machine learning; these would likewise need to be capable of interpreting bit strings as rules or as rule sets. However, all other operational aspects of genetic search are confined to the manipulation of bit strings as bits, without regard to any meaning which may be assigned to them externally.

To reflect this duality, in the descriptions which follow, the term 'individual' will be used when referring to the bit string itself, and the term 'rule' or 'rule set', as appropriate, will be used when referring to the interpretation of the bit string.

4.4 Specific Applications of the Genetic Algorithm to Machine Learning

This section describes three specific implementations of genetic learning.

4.4.1 McCallum and Spackman's Algorithm

Unlike most learning programs based upon the Michigan approach, the unnamed implementation described in [McCallum90] does not involve a classifier system. Instead, this particular program is noteworthy both for the method it uses to construct a rule set from the members of its population, and for a novel means of increasing the exploration rate of its genetic search.

4.4.1.1 Example Sharing

The primary difficulty of the Michigan approach is that, by nature, the population of rules would normally converge toward a single rule which correctly classifies the greatest number of training instances; however, many concepts cannot be expressed properly with only one rule. Thus, the classical genetic algorithm must be modified so that it will produce as many rules as might be required.

In general, a technique called *niche formation* [Goldberg89, p. 187] is used in genetic algorithms when the search space contains multiple peaks, *i.e.* multiple local optima, and the goal is to find all of these peaks rather than concentrating only on the highest one. Typically this is done by modifying the fitness function applied to individuals in such a way as to make the result proportional to the number of individuals in each (arbitrarily defined) 'neighbourhood'. A common niche formation technique is called *payoff sharing*, which involves dividing the absolute fitness value of each individual in a given neighbourhood by the number of individuals in that neighbourhood.

McCallum and Spackman perform niche formation by means of a modification of payoff sharing which they call "example sharing". The idea is that every training instance has a finite payoff value which is shared among those rules which cover that instance; intuitively, one can "think of each example as having only a finite amount of payoff to give, and this finite payoff being shared equally among its recipients" [McCallum90, p. 150]. The payoff value is used in the definition of the fitness function, as follows:

- Every positive training instance e carries a total payoff value $V_e = 1.0$.
- Every negative training instance e carries a total payoff value $V_e = -1.0 \cdot s$, where s is a user-supplied constant with a default value of 1.0; the greater the value of s, the more specific the resulting rules will be.
- The payoff value which an individual rule i receives from a training instance e is denoted P_{ie}, and defined as

$$P_{ie} = \begin{cases} 0, & \text{if i doesn't cover } e \\ \frac{V_e}{c_e}, & \text{if i covers } e \end{cases}$$

  where c_e is the number of rules which cover instance e.

- The total payoff which a rule receives from all instances in training set T that it covers is then

$$t(i) = \sum_{e \epsilon T} P_{ie}$$

- The fitness function is defined as

$$f(i) = \begin{cases} t(i) + 1, & \text{if } t(i) \geq 0 \\ e^{t(i)}, & \text{if } t(i) < 0 \end{cases}$$

The effect of this fitness function is to divide the population into portions which converge, simultaneously and independently, toward different, cooperating rules. The resulting rules cover disjoint subsets of the training set, and their union forms the newly learned concept.

4.4.1.2 Overpopulation

One difficulty with the classical genetic algorithm is the requirement of assigning probabilities to the application of the crossover and mutation operators; the optimal values for a particular problem domain are often not known in advance, yet an inappropriate choice might slow convergence, or cause premature convergence to a local optimum.

This problem can be avoided using a mechanism that McCallum and Spackman term *overpopulation*, which can be expressed as a modified version of the classical genetic algorithm, as shown in Figure 10.

1. Let time t = 0.
2. Initialize population P(t).
3. Copy all individuals from P(t) to temporary holding pool T.
4. Apply crossover operator to all individuals currently in T, with probability 1.0, appending the newly created individuals to T.
5. Apply mutation operator to all individuals currently in T, with probability 1/n (where n is the number of bits in each individual), appending the newly created individuals to T.
6. Compute fitness for each individual in T.
7. Let t = t + 1.
8. Select new population P(t) from individuals in T (refer to Section 4.2.2 for a discussion of how this may be done).
9. Repeat steps 3 - 8 until some arbitrary termination criterion is reached.

Figure 10: The genetic algorithm with overpopulation

In other words, the temporary holding pool includes all unmodified members of the current generation. Crossover is applied with probability 1.0, and the children added to the temporary holding pool. Mutation is then applied with probability 1/n (where n is the number of bits in each individual), both to the original members of the current generation and to the children created by the crossover operator.

The temporary holding pool thus doubles at each step, resulting in a total size of $p \cdot 2^n$, where p is the number of individuals in the population and n is the number of operators applied (typically $n = 2$, namely crossover and mutation, as in the case of [McCallum90]; however, the method can be applied with additional operators in the general case). However, the next generation is formed by selecting only p individuals from the temporary holding pool.

In addition to avoiding the problem of having to select operator probabilities, overpopulation increases the rate of exploration of the genetic search. The method increases the number of individuals which must be evaluated in each generation, but decreases the number of generations required to converge to a solution [McCallum90, p. 151].

Overpopulation appears not to be widely used by other researchers, although the "Elitist Selection" method in Eshelman's CHC algorithm [Eshelman91] is similar.

4.4.2 GABIL

A challenge faced by genetic learners which represent concepts in the form of rule sets is the fact that not all rule sets contain, or should contain, the same number of rules. Some concepts can be adequately represented by a single rule, while others may require dozens or hundreds of rules to be represented accurately.

The Michigan approach generally deals with this issue by some form of niche formation, as in [McCallum90]. The Pittsburgh approach, however, encodes an entire rule set in each individual, which implies either that all rule sets must contain an identical number of rules, or that the genetic algorithm must be extended to function in the case where individuals contain bit strings of variable length. GABIL ("Genetic Algorithm Batch Incremental Learner") [De Jong93] is an example of a concept-learning program which combines the Pittsburgh approach with variable-length bit strings.

4.4.2.1 Representation Format

GABIL represents concepts as rules which are almost but not quite in k-DNF form (Section 1.1.3.2). In particular, each rule contains one conjunct for each training attribute, and each conjunct specifies one or more values for the given attribute which a member of the target concept may contain.

For example, suppose the concept WIDGET is described by Rule Set 4.1, obtained from the attributes shown in Table 21.

Attribute	Possible Values of the Attribute
size	small, medium, large
colour	red, orange, yellow, green, blue
shape	circle, square

Table 21: Attributes which describe objects that might be widgets

Rule Set 4.1

```
(size = medium) → widget
(colour = yellow) ∧ (shape = circle) → widget
(colour = red ∨ green) ∧ (shape = square) → widget
(size = large) ∧ (colour = orange ∨ green) ∧ (shape = square) → widget
```

The difference between this format and k-DNF is that conjuncts are permitted to contain more than one value for the given attribute (*e.g.* `colour = red ∨ green`); this is called **internal disjunction**, and [De Jong93, p. 164] cites [Michalski83] as justification for using it.

As with Rule Set 1.5, any instance which satisfies any one of the rules in Rule Set 4.1 is considered to be a member of the target concept, while any instance which satisfies none of the rules in the rule set is considered not to belong to the concept.

4.4.2.1.1 Attributes

GABIL only deals with discrete attributes, and thus it represents an attribute as a string of bits, with one bit for every possible value it may assume. Within the string, bits set to one denote values for the given attribute which are permitted in instances which are members of the target concept, and bits set to zero denote values for the attribute which are not permitted. For example, the **colour** attribute of Table 21 has five possible values, and so is represented by five bits; Table 22 shows some sample bit strings and their interpretations.

Note that according to this convention, a bit string in which all bits are set to one corresponds to an attribute for which any value is acceptable. Intuitively, this is tantamount to dismissing that attribute as being irrelevant, such as the **Age** attribute in the 'lion example' of Section 1.1.3.1. The opposite extreme is a string in which all bits are zero; a rule containing such a string can never be satisfied by any instance.

Attribute Bit String	Interpretation
10000	red
11000	red or orange
01110	orange or yellow or green
00001	blue
11111	any value at all
00000	**no** value is acceptable

Table 22: Sample bit strings for the **colour** attribute

4.4.2.1.2 Rules

A rule is represented by the concatenation of the **conjuncts** (*i.e.* attributes) it contains. For example, the rule

(size = large) ∧ (colour = orange ∨ green) ∧ (shape = square) → widget

would be represented by the bit string

```
001 01010 01
```

(the spaces in the bit string, which mark attribute boundaries, are shown for clarity but are not actually part of the format).

Note that the right-hand side (**consequent**) of the rule is not included in this representation. This is possible because GABIL only attempts to learn a single concept at a time [De Jong93, p. 166], which means that all rules necessarily have the same consequent (*i.e.* membership in the target concept); thus, the consequent is inferred but does not need to be represented explicitly.

4.4.2.1.3 Rule Sets

A rule set is represented as the concatenation of one or more rules. Thus, since each individual in GABIL's population is a complete rule set, the number of bits in the representation of a rule set is not fixed. However, there are limits on the extent to which the length can vary, since any bit string must contain an integral multiple of the number of bits required to represent a single rule, and the length of a rule is in fact constant for any given data set.

The rules in a rule set are not interpreted in any particular order; an instance which satisfies any one rule satisfies the rule set as a whole, regardless of which rule it might be or of whether it might satisfy more than one rule.

As an example, recall Rule Set 4.1, reproduced here for convenience:

Rule Set 4.1

(size = medium) → widget
(colour = yellow) ∧ (shape = circle) → widget
(colour = red ∨ green) ∧ (shape = square) → widget
(size = large) ∧ (colour = orange ∨ green) ∧ (shape = square) → widget

GABIL's representation of this rule set would be as follows:

```
010 11111 11 • 111 00100 10 • 111 10010 01 • 001 01010 01
```

The spaces marking attribute boundaries and bullets ('•') marking rule boundaries are shown only for visual clarity, and are not part of the format.

4.4.2.1.4 The Michigan Approach

As an aside, at this point the distinction between the Pittsburgh and Michigan approaches can be made clear by example. Specifically, since the Michigan approach considers each individual in the population to be a single rule, a learning system based on it would encode the above rule set as

```
010 11111 11
111 00100 10
111 10010 01
001 01010 01
```

where each 10-bit string is an independent individual, and represents a single rule. Of course, a population composed in this way will also contain other rules which may be less relevant, but which nevertheless would be included in the final concept description unless a method is found to detect and remove them. The difficulty in doing so may be one reason why the Pittsburgh approach is favoured by some researchers.

4.4.2.2 Coping With Variable-Length Bit Strings

The classical genetic algorithm is defined for individuals with bit strings of constant length. However, the only part of the algorithm which is actually sensitive to variations in bit string length is the crossover operator; the selection and mutation steps function identically with variable-length bit strings as they do with fixed-length strings.

GABIL uses a modified version of two-point crossover. In the standard implementation of two-point crossover, two points are chosen at random, and the bits between those points are exchanged between two individuals. When the individuals in question are not guaranteed to be the same length, it might happen that one or both of the selected points may not exist in the shorter of the two individuals. GABIL handles this by guaranteeing that the crossover points in the two individuals "match up semantically" [De Jong93, p. 166], *i.e.* that the corresponding crossover points occupy the same position relative to a rule boundary. As an example, consider the following two rule sets:

```
111 11111 11 • 111 11111 11 • 111 11111 11 • 111 11111 11
000 00000 00 • 000 00000 00
```

Each rule contains ten bits, with four rules in the first rule set and two in the second. Suppose for the sake of discussion that the randomly chosen crossover points in the first rule set are 5 and 36, so that bits in the inclusive range 6 – 35 will be exchanged. Since bits are numbered from zero, starting at the left end of the string, this means that the range of bits to be exchanged from the first rule set is as follows:

```
111 11111 11 • 111 11111 11 • 111 11111 11 • 111 11111 11
```

Position 5 exists in the second rule set, but 36 does not, meaning that a point congruent to it must be found. Since position 36 follows the sixth bit of a rule, and rules are ten bits long, congruent positions include 6, 16, 26 and 36; of these, only 6 and 16 are available in this particular case. If 16 is selected from these two at random, then the ranges of bits to be exchanged in both rule sets are as follows:

```
111 11111 11 • 111 11111 11 • 111 11111 11 • 111 11111 11
000 00000 00 • 000 00000 00
```

The result of this particular crossover would then be

```
111 11111 11 • 111 11100 00 • 000 00011 11
000 00011 11 • 111 11111 11 • 111 11100 00
```

This example illustrates a number of points:

- The number of rules in each rule set has changed, but the total number of rules in both rule sets is conserved. The latter point will always be true, but the former depends on the particular crossover points which are chosen; it can certainly happen that both offspring will have the same length as their parents.

- By forcing the crossover points in both rule sets to be congruent with respect to rule boundaries, both offspring are guaranteed to contain a bit string of appropriate length, *i.e.* an integral multiple of the number of bits per rule.

- Either crossover point may occur in the middle of an attribute. When this happens, each child receives a new string for the attribute in question; this is not only allowable but desirable, since it provides the only means other than mutation by which new attribute strings may be generated.

Finally, note that in two-point crossover, the bit string is considered to be circular. For example, given the same two rule sets, suppose that the cross points are 15 and 36 in the first rule set, and 15 and 6 in the second. The range of bits to be exchanged from the second rule set would then be bits number 16 – 19 and 0 – 5 inclusive.

```
111 11111 11 • 111 11111 11 • 111 11111 11 • 111 11111 11
000 00000 00 • 000 00000 00
```

The resulting offspring would be

```
111 11111 11 • 111 11100 00 • 000 00011 11
111 11100 00 • 000 00011 11 • 111 11111 11
```

4.4.2.3 Genetic Operators for Learning

The classical genetic algorithm makes no use of domain knowledge even if it is available; this is the trade-off it makes in exchange for being applicable to any domain that can be represented in an appropriate form [Goldberg89]. However, there is no reason why the algorithm might not be modified to improve its performance in a particular domain, such as machine learning.

One way to extend the algorithm is to define new operators, specialized for a particular task. GABIL in fact defines two such operators, which the authors term *adding alternative* and *dropping condition*. Both of these are based on operators of the same name reported in [Michalski83].

4.4.2.3.1 The *Adding Alternative* Operator

The adding alternative operator generalizes a rule by adding an internal disjunct. For example, consider a rule such as

$$(\texttt{colour} = \texttt{orange} \vee \texttt{green}) \wedge (\texttt{shape} = \texttt{square}) \rightarrow \texttt{widget}$$

The adding alternative operator would add a new acceptable value for one of the attributes, potentially resulting in

$$(\texttt{colour} = \texttt{red} \vee \texttt{orange} \vee \texttt{green}) \wedge (\texttt{shape} = \texttt{square}) \rightarrow \texttt{widget}$$

or

$$(\texttt{colour} = \texttt{orange} \vee \texttt{green}) \wedge (\texttt{shape} = \texttt{circle} \vee \texttt{square}) \rightarrow \texttt{widget}$$

As used in GABIL, the disjunct to be added is chosen at random. The operator is implemented as an asymmetric variation of ordinary mutation, which has a probability of 0.75 that a zero bit will be inverted to a one, but only a 0.25 probability that a one bit will be inverted to a zero [De Jong93, p. 176]. Note that this is employed in addition to rather than instead of standard mutation, but with ten times the probability of application (0.01 vs. 0.001).

4.4.2.3.2 The *Dropping Condition* Operator

The dropping condition operator generalizes a rule by removing a conjunct entirely if it appears to be almost irrelevant. As implemented in GABIL, this applies to any

attribute string in a rule which has more than half of its bits already set to one, in which case the remaining zero bits are changed to one, thus effectively removing any reference to that attribute in the given rule. As an example, consider the rule

(colour = red ∨ orange ∨ green) ∧ (shape = square) → widget

This would be represented by the bit string

```
111 11010 01
```

In this case, only the **colour** attribute has more than half its bits set to one. It would thus be changed to all ones, resulting in a bit string of

```
111 11111 01
```

which translates back to

(shape = square) → widget

The adding alternative operator is similar to mutation in that its effect is essentially random. In contrast, if the dropping condition operator applies at all (subject to a probability of 0.6, identical to that of crossover), its effect will be deterministic — rather than inverting a single bit at random, it will always change all zero bits to one within a given attribute.

4.4.2.3.3 Incorporating New Operators into the Algorithm

The order in which the operators are invoked in GABIL is

1. crossover
2. mutation
3. adding alternative
4. dropping condition

Control flow in GABIL differs from that of the classical genetic algorithm (Section 4.2.1) only in that two extra steps are added.

4.4.2.3.4 Interaction Between Operators

The adding alternative and dropping condition operators are not mutually independent. In particular, the adding alternative operator is capable of adding enough internal disjuncts to make an attribute suitable for removal by dropping condition. The question of whether or not this should be allowed to occur, along with the question of how to set the most effective possible probability of application for each operator, motivated one of GABIL's more interesting features, which is described in Section 4.4.2.4.

4.4.2.4 Self-Adaptation

As described in Section 4.2.5, most implementations of the genetic algorithm offer a number of adjustable parameters which affect how the search will be performed; this holds true of genetic learning applications as well. Often even an experienced user may find it difficult to select appropriate values for these parameters without a great deal of trial and error, and as a result attempts have been made to modify the genetic algorithm to adapt itself automatically [Grefenstette86, Schaffer87, Davis89, Janikow91], a process known as **self-adaptation**.

In GABIL, the parameters which are handled in this way are the application probabilities of the two new learning-specific operators, adding alternative and dropping condition. Specifically, two *control bits* are added to every individual in the population, one for each of the two operators. These bits act as "added Boolean preconditions for the operators" [De Jong93, p. 180]. For example, when an individual is selected from the mating pool for application of the adding alternative operator, the control bit for that operator is evaluated. If it is set to one, then the operator will be applied in the usual way subject to its normal probability; if it is set to zero, then the operator will not be applied (*i.e.* the individual will be passed on unchanged) regardless of its probability. The dropping condition operator is handled similarly.

What makes this scheme adaptive is that the control bits are subject to manipulation by selection, crossover and mutation, although not by the adding alternative or dropping condition operators themselves. Thus, while searching for the best possible rule set, GABIL simultaneously attempts to determine for itself whether to apply its learning-specific operators to any given rule set. This provides a means by which the algorithm dynamically adjusts its bias (Section 1.1.3.2).

4.4.2.5 Fitness

The fitness function used in GABIL is the square of the percentage of training instances which are correctly classified by an individual rule set. This choice encodes a bias in favour of rule sets which are accurate, while deliberately not attempting to influence the length or complexity of the learned concept. The stated reason for squaring the accuracy is to provide "a bias toward correctly classifying all the examples while providing a non-linear differential reward for imperfect rule sets" [De Jong93, p. 167].

4.4.2.6 Test Results

In [De Jong93], results were published for five different versions of GABIL:

- the original version, without either of the two new operators (adding alternative and dropping condition)
- with adding alternative but without dropping condition
- with dropping condition but without adding alternative
- with both new operators, but without self-adaptation
- with both new operators, using self-adaptation

De Jong *et al* tested all five variants of the algorithm using the Ljubljana breast cancer data set (Section 2.3), with results as shown in Table 23. The column labelled 'Accuracy' lists the classification accuracy reported in [De Jong93] for the Ljubljana data set. Note, however, that due to the use of an instance presentation method that [De Jong93] terms *batch-incremental mode*, the entire data set was used both for training and for recall [De Jong93, p. 168], as opposed to the split into training and recall sets described in Section 2.3 and used for all other tests reported in this thesis.

The baseline accuracy for the complete Ljubljana data set is 70.3% (based on a total of 286 instances, of which 201 are positive). Thus, only GABIL+D and GABIL+AD were able to exceed the baseline accuracy for this data set. It is interesting to note that the adaptive version (*i.e.* Adaptive GABIL) did not perform as well as the version in which both operators were applied without self-adaptation (*i.e.* GABIL+AD); the authors hypothesize that this is due to the relatively small

Algorithm Variant	Accuracy
GABIL (original)	68.7%
GABIL+A (with adding alternative)	69.1%
GABIL+D (with dropping condition)	71.5%
GABIL+AD (with both new operators)	72.0%
Adaptive GABIL	70.3%

Table 23: Classification accuracy for GABIL on the Ljubljana data set

population size used in that experiment, and propose to adapt population size as well as operator selection in future versions [De Jong93, pp. 181–182].

[De Jong93] also reports on a series of experiments using a series of artificial concepts. In this series, Adaptive GABIL produced the best result among the five algorithm variants, GABIL+D was second best, and GABIL+AD was third. The fact that GABIL+AD performs less well than both GABIL+D and the adaptive version on the artificial concepts "shows the danger of indiscriminately including multiple fixed biases, which can interfere with each other, producing lower performance" [De Jong93, p. 181]; in other words, new learning-specific operators should not be introduced indiscriminately, without considering the effect they might have on other operators, or without some means of mitigating such side effects as do occur. This is the reason why self-adaptation was added to GABIL [De Jong93, pp. 178–179], although self-adaptation is not the only means by which this effect might be handled (*e.g.* see Section 4.4.3.3.4).

4.4.3 The Proposed ELGAR Algorithm

ELGAR ('Evolutionary Learning by Genetic AlgoRithm') is a genetic learning system which was written to evaluate the effectiveness of different genetic learning techniques. The resulting algorithm was then combined with ideas from Holte's 1R classifier [Holte93], in order to create a hybrid algorithm with which to evaluate the ideas proposed in this thesis. The genetic component of the system (*i.e.* ELGAR) will be described here, while the hybrid system (called 'Hybrid ELGAR' in order to distinguish it from the original) will be presented in Chapter 5.

4.4.3.1 Data Representation in ELGAR

ELGAR represents rule sets in the same format as that of GABIL (see Section 4.4.2.1), and thus, like GABIL, is an example of the Pittsburgh approach to genetic learning. The only respect in which ELGAR's representation format differs from GABIL's is that ELGAR does not perform self-adaptation (see Section 4.4.3.3.4), and thus does not require or use control bits.

4.4.3.2 Operators

ELGAR currently implements the operators listed in Table 24. The user may choose to apply any subset of these operators in any given run of the program.

Operator Type	Name	Reference	Type
one-point crossover	cross1	[Goldberg89]	binary
two-point crossover	cross2	[Goldberg89]	binary
uniform crossover	crossU	[Goldberg89]	binary
modified uniform crossover	crossMU	[Eshelman91]	binary
rule crossover	crossR	[Michalewicz92]	binary
mutation	mutation	[Goldberg89]	unary
adding alternative	addalt	[De Jong93]	unary
dropping condition	dropcond	[De Jong93]	unary
generalization/specialization	genspec	[Michalewicz92]	unary

Table 24: Operators implemented in ELGAR

4.4.3.2.1 One-point and Two-point Crossover

These operators are described in Section 4.2.4. Since ELGAR follows GABIL's variable-length data representation, it also borrows GABIL's implementation of two-point crossover [De Jong93] (see Section 4.4.2.2), and extends one-point crossover in exactly the same fashion: crossover points are randomly selected in the longer of the two parent individuals, and forced to be congruent with respect to rule boundaries in the shorter parent.

4.4.3.2.2 Uniform Crossover

This operator is described in Section 4.2.4. In ELGAR, uniform exchange of bits occurs only in those bit positions which exist in both parents; the remaining portion

of the longer parent is simply copied to the longer child. Note that this implies that uniform crossover will always create children with the same lengths as their parents. As an example, consider the following two rule sets:

```
111 11111 11 • 111 11111 11
000 00000 00
```

Each rule contains ten bits, with two rules in the first rule set and one in the second. Uniform crossover might produce the following offspring

```
010 01000 10 • 111 11111 11
101 10111 01
```

4.4.3.2.3 Modified Uniform Crossover

This operator is described in Section 4.2.4. The implementation in ELGAR, like that of uniform crossover, considers only those bit positions which are common to both parents, and copies the remaining portion of the longer parent to the longer child. Among those bit positions which exist in both parents, those which differ in value between the two parents are exchanged between the two children such that each child receives half the differing bits from each parent. For example, given the two rule sets

```
111 11111 11 • 111 11111 11
000 01111 00
```

the resulting children might be

```
100 11111 01 • 111 11111 11
011 01111 10
```

4.4.3.2.4 Rule Crossover

This operator is a variation of uniform crossover which creates two children by randomly exchanging entire rules from the two parents. This is inspired by the "RuleExchange" operator used in GIL [Janikow91] and described in [Michalewicz92, p. 225]. For example, given the two rule sets

```
111 11111 11 • 111 11111 11 • 111 11111 11 • 111 11111 11
000 00000 00 • 000 00000 00 • 000 00000 00 • 000 00000 00
```

the resulting children might be

```
111 11111 11 • 000 00000 00 • 111 11111 11 • 000 00000 00
000 00000 00 • 111 11111 11 • 000 00000 00 • 111 11111 11
```

Note that rule crossover will never create any new rules; it will only exchange rules between rule sets. For this reason, it should be used in conjunction with at least one other form of crossover.

4.4.3.2.5 Mutation

This operator is described in Section 4.2.3. The mutation operator in ELGAR always flips exactly one randomly chosen bit in every parent individual. This is a variation on the mutation operator defined for use with overpopulation by McCallum and Spackman, which applies with a probability of $1/n$, where n is the number of bits per individual [McCallum90]. While this is substantially greater than the usual probability used for mutation (commonly 0.001, according to [Goldberg89]), nevertheless it retains the possibility for a given mutation to be ineffective, which is redundant in the context of overpopulation. Thus, ELGAR guarantees that a mutated version of every parent individual will be added to the temporary holding pool to be considered for selection for the next generation.

4.4.3.2.6 Adding Alternative

This operator is implemented as in [De Jong93] (see Section 4.4.2.3.1), with the difference that it will never change a bit value from one to zero. The reason for this decision is that zero-to-one mutations are already handled by the regular mutation operator, and the mandate of adding alternative is to generalize rather than specialize.

4.4.3.2.7 Dropping Condition

This operator is implemented exactly as in [De Jong93] (see Section 4.4.2.3.2).

4.4.3.2.8 Generalization/Specialization

The genspec operator is effectively a combination of the "NewPEvent" and "NewNEvent" operators used in GIL [Janikow91] and described in [Michalewicz92, pp. 225–227]. Given a rule set and an instance which is not correctly classified by that rule

set, genspec will modify the rule set so as to classify the instance correctly. As used in ELGAR, genspec always operates on the lowest-numbered instance in the training set which is misclassified by the rule set.

In the case of an uncovered positive instance, genspec will add a rule which covers that one instance explicitly. For example, consider the following rule set, using the WIDGET concept and attributes described in Table 21:

```
(colour = red ∨ orange ∨ green) ∧ (shape = square) → widget
(colour = blue ) ∧ (shape = square) → widget
```

This would be represented by the bit string

```
111 11010 01 • 111 00001 01
```

Suppose that an instance describing an object which is small, green and circular is found to be a widget; in that case, the result of genspec would be

```
111 11010 01 • 111 00001 01 • 100 00010 10
```

or

```
(colour = red ∨ orange ∨ green) ∧ (shape = square) → widget
(colour = blue ) ∧ (shape = square) → widget
(size = small) ∧ (colour = green) ∧ (shape = circle) → widget
```

A covered negative instance is handled by randomly selecting an attribute value required by that instance, and removing that value from all rules in the rule set. For example, given the rule set

```
(colour = red ∨ orange ∨ green) ∧ (shape = square) → widget
(colour = yellow ∨ blue ) ∧ (shape = square) → widget
(colour = orange ∨ green ) ∧ (shape = circle) → widget
```

or

```
111 11010 01 • 111 00101 01 • 111 01010 10
```

suppose we find an orange, square object which is not actually a widget. In this case genspec will randomly select an attribute; for the sake of illustration, suppose that the colour attribute is chosen. Then the attribute value to be removed would be orange, and the resulting child individual would be

```
111 10010 01 • 111 00101 01 • 111 00010 10
```

or

```
(colour = red ∨ green) ∧ (shape = square) → widget
(colour = yellow ∨ blue ) ∧ (shape = square) → widget
(colour = green ) ∧ (shape = circle) → widget
```

Note that the given attribute value must be removed from all rules in the rule set in order for the result to be effective.

The implementation of NewNEvent in GIL [Janikow91] is considerably more sophisticated, in that it will take much more care to ensure that the resulting child will cover exactly the same set of instances that it did before, with the sole exception of the negative instance which is known to be covered incorrectly. However, this is accomplished by adding new rules, the number of which will depend on how many rules in the rule set cover the negative instance, and how many attributes in each such rule have more than one bit set to one. In practice, this tends to cause offspring to grow to a length which makes them awkward to manipulate. This is why the GIL implementation is not used in ELGAR.

One potential weakness of combining both NewPEvent and NewNEvent into a single operator is that it might be possible for the combined operator to cause a given rule set to oscillate between two forms. For example, suppose a rule set R does not cover a positive training instance T_1. The genspec operator would generalize R to make it cover T_1, creating a new rule set R'. Suppose that R' now covers a negative training instance T_2 which had not been covered by R. It may now happen that rule set R'', created by applying genspec to specialize R', will not cover T_2, but will also no longer cover T_1. In practice there are two reasons why this is unlikely to be a problem, however:

- In the generalization case as described above, a rule set such as R' is created by adding a specific rule to the parent rule set (*i.e.* R in the above example). The new rule in question is explicitly constructed to match the specific positive instance which had not previously been covered. Consider the instances which will be covered by R' but not covered by R; clearly these will be identical to the original training instance T_1, except possibly for the classification value. Thus, the only way in which R' can cover a negative instance not covered by R is if

the negative instance is otherwise identical to positive instance T_1 — but this can only happen if the training set is inconsistent.

- Unless both versions of a given rule set (*e.g.* R and R′, or R′ and R″) are equally fit, only the fitter of the two is likely to be retained in the next generation. This won't stop the second version from being created, but at least it won't waste space in the next generation for a rule set which is known to be no better than one which existed in a previous generation.

4.4.3.3 A Modified Genetic Algorithm

ELGAR offers the user a choice between two versions of overpopulation: the original as defined by McCallum and Spackman[McCallum90], or a modified version which sacrifices some exploratory power in favour of reduced memory and CPU consumption. The main processing loop in ELGAR is thus similar to that of McCallum and Spackman's approach (Section 4.4.1.2) — in fact, it will be identical if the McCallum and Spackman version of overpopulation is used, and if the only two operators applied are crossover and mutation. The main loop in ELGAR with the McCallum and Spackman version of overpopulation is as shown in Figure 11.

---

1. Let time t = 0.
2. Initialize population P(t).
3. Copy all individuals from P(t) to temporary holding pool T.
4. For every available operator which the user has chosen to apply:
   (a) Apply the operator to all individuals currently in T, with probability 1.0, appending the newly created individuals to T.
   (b) Compute fitness for all individuals just added to T.
5. Let t = t + 1.
6. Select new population P(t) from individuals in T.
7. Repeat steps 3 - 6 until a user-selected termination criterion is reached.

Figure 11: The main loop in ELGAR with 'real' overpopulation

4.4.3.3.1 Termination Criteria

If a rule set is found which correctly classifies all available training instances, ELGAR's genetic search will automatically terminate at the end of the generation which produced that rule set. Otherwise, the user may opt to test for any or all of the following criteria, causing ELGAR's genetic search to terminate as soon as the first selected condition occurs.

- a user-specified maximum number of generations
- *convergence*, which is defined in ELGAR as the situation in which the average fitness of the current generation reaches 98% (an arbitrarily chosen value) of the fitness of the most highly fit individual seen to date; the assumption is that continuing the search past this point would be unproductive
- *stagnation*, which is defined in ELGAR as a lack of improvement over a user-specified number of consecutive generations (where 'improvement' is defined as any increase in the fitness of the most highly fit individual seen to date)

The convergence criterion is intended to monitor the condition that the population has become dominated by a single highly fit individual. When this happens, continuing exploitation via crossover is unlikely to be helpful, since a great majority of individuals are either identical or highly similar to each other. At that point, only mutation remains as a viable means of generating new individuals, and this process is sufficiently slow that, for all intents and purposes, the search is no longer effective.

The stagnation criterion is a pragmatic means of ending a search which no longer appears to be productive. The number of generations of tolerance can be selected by the user; a value of zero causes ELGAR to ignore this criterion altogether.

4.4.3.3.2 Modified Overpopulation

Due to the memory requirements imposed by the McCallum and Spackman version of overpopulation, particularly when many operators are used, ELGAR includes a modified form of overpopulation in which operators are not applied to the newly created individuals in the temporary holding pool, but only to the original members of the current generation. The algorithm for this is as shown in Figure 12:

1. Let time t = 0.
2. Initialize population P(t).
3. Copy all individuals from P(t) to temporary holding pool T.
4. For every available operator which the user has chosen to apply:
   (a) Apply the operator to all members of P(t), with probability 1.0, appending the newly created individuals to T.
   (b) Compute fitness for all individuals just added to T.
5. Let t = t + 1.
6. Select new population P(t) from individuals in T.
7. Repeat steps 3 - 6 until a user-selected termination criterion is reached.

Figure 12: The main loop in ELGAR with modified overpopulation

If p is the number of individuals in the population and n is the number of operators applied, then 'real' overpopulation as defined by McCallum and Spackman would create a temporary holding pool containing a total of $p \cdot 2^n$ individuals (see Section 4.4.1.2). In contrast, the number of individuals created by the modified form of overpopulation is only $p \cdot (n + 1)$.

This has been shown experimentally (see Section 4.5.2) to provide enough exploration to be useful, while still requiring significantly less memory and less processing time than the McCallum and Spackman version of overpopulation.

4.4.3.3.3 Overpopulation vs. Operator Application Probabilities

When either form of overpopulation is used, the original members of any given generation are included without modification in the temporary holding pool, along with their offspring as generated by whichever operators are applied. As a result, the concept of operator application probability has no meaning (or, alternatively, all operators are always applied with a probability of 1.0), since there is no longer any reason why it might be useful for an operator not to replace a parent individual by its offspring.

Justification for this policy is based on the original concept behind overpopulation, as enunciated by McCallum and Spackman [McCallum90, p. 151]:

> Intuitively one can imagine a situation where the birthrate far exceeds the sustaining support of the environment, and individuals are dying off not from old age, but from competition with each other — thus, we say over-population.

Thus, all parents and all of their offspring are allowed to compete amongst themselves simultaneously in every generation, and the problem of the probability with which to apply any given operator never arises.

4.4.3.3.4 Overpopulation vs. Self-Adaptation

The same argument which justifies the unimportance of operator application probabilities also applies to self-adaptation of operator applicability. While the self-adaptation method in GABIL (see Section 4.4.2.4) is both powerful and elegant, the use of overpopulation renders it redundant.

4.4.3.4 Selection Method

ELGAR uses binary tournament selection, with a small probability of allowing the individual with lower fitness to be the tournament winner. The probability value used for this purpose decreases over time, in a sequence of ten equal-sized steps beginning at 0.3 and ending at 0.05. The particular value used at any given moment is selected based on how close the current generation is to the maximum allowable number of generations, according to the equation

$$\text{step number} = \min(\lfloor 10 \times (\text{generation number} \div \text{generation limit}) \rfloor, 10)$$

The reason for using tournament selection is that it is much simpler computationally than most other methods and has the advantage of not requiring fitness values to be scaled [Goldberg89], while simultaneously performing as well as other commonly used techniques [Miller95], [Goldberg91, p. 90].

The reason for starting with a high probability of reversing the tournament result, and then decreasing this probability over time, is to increase selection pressure

over time. Specifically, in the early stages of the search there will be a relatively large number of diverse individuals in the population, with a correspondingly greater likelihood that an individual of lower fitness might actually be more useful simply because it is different and might therefore contain a building block that would be usefully combined with those in more highly fit individuals. However, when the population has begun to converge, the more likely it becomes that fitness really will correlate with actual worth, and thus it becomes less desirable to risk losing a more highly fit individual in favour of one which is less fit.

4.4.3.5 Fitness Criteria

Fitness in ELGAR consists of a base value computed by a specific fitness function, with optional bonuses or penalties applied according to criteria selectable by the user.

4.4.3.5.1 Fitness Functions

A number of different fitness functions were evaluated during the development of ELGAR. The one currently used is based on that of GABIL, namely the square of the percentage of training instances correctly classified. However, all of the functions listed in Table 25 have been tried, with results as described in Section 4.5.2.

Fitness Function	Reference
p^2	[De Jong93]
p	
$10 \cdot p$	
p^3	
e^p	
$1 - \textit{Laplace Error}$	[Quinlan95]
$\textit{completeness} \cdot (1 - \textit{consistency})$	[Zhang94]

Table 25: Fitness functions tested in ELGAR

In Table 25, the value p refers to the percentage of training instances correctly classified by the given rule set. A rule set's *Laplace error* [Quinlan95] is computed as

$$\mathcal{L}(n, e) = \frac{e + k - 1}{n + k}$$

where k is the number of classes to be learned, n is the number of training instances covered by the rule set, and e is the number of false positives, *i.e.* negative instances

which are covered by the rule set. When $k = 2$, as is the case with ELGAR (in which the two classes are **member of the target concept** and **non-member of the target concept**), Laplace error reduces to

$$\mathcal{L}(n, e) = \frac{e + 2 - 1}{n + 2} = \frac{e + 1}{n + 2}$$

Note that genetic search implicitly maximizes fitness, but the Laplace error must be minimized in order to produce the best possible concept description; this is why the Laplace error is subtracted from 1.0.

Similarly, *completeness* is defined as the percentage of positive instances which are correctly classified, and *consistency* (which, in this context, should perhaps have been termed 'inconsistency' instead) is the percentage of false positives [Zhang94, pp. 442–443]. The goal is to maximize completeness while simultaneously minimizing the percentage of false positives, which is why the fitness function involving these terms is written as shown in Table 25.

The reason for testing functions such as p, $10 \cdot p$, p^3 and e^p was to see how they compare to the p^2 function used in GABIL. As shown in Section 4.5.2, ELGAR produces identical results using any of these functions, given that all other operating parameters are unchanged. In hindsight this is not surprising, due to the influence of binary tournament selection. This selection scheme relies only on the direction of the inequality between two fitness values, rather than on the magnitude of their difference; for example, a tournament involving fitness values of 0.001 and 0.001001 would produce the same outcome as one involving values of 1,000,000 and 100. It seems likely that, had other selection schemes been used instead, this result may well have been different.

4.4.3.5.2 Fitness Tuning

In addition to the basic fitness computation, the user may choose to influence ELGAR by applying either or both of the following:

- a bonus equal to ten times the amount by which a rule set's accuracy exceeds the baseline accuracy of the training set
- a penalty equal to 0.001 multiplied by the number of rules in the rule set

The first of these items is intended to reward rule sets which perform better than random guessing, while the second is intended to break ties in favour of shorter rule sets. Note that the fitness value will never be negative under any circumstances, with values that would otherwise be negative being replaced by zero.

4.5 Experimental Results

4.5.1 GABIL

The only data set among those considered here for which [De Jong93] reports an experimental result is the Ljubljana breast cancer data, for which GABIL+AD obtained a classification accuracy of 72.0% (see Table 23).

4.5.2 ELGAR

Twenty experiments were conducted using ELGAR. All runs were performed on a 166 Mhz Intel Pentium system with 32 megabytes of memory, running the Slackware 3.2 distribution of Linux with kernel 2.0.30. ELGAR is written in ANSI C, and was compiled with gcc version 2.7.2.

In order to minimize the effects of the randomness involved in genetic search, every experiment performed five complete genetic searches, each in a separate pass, using a single continuous sequence of random numbers generated by the Linux implementation of the random() function from the standard C library. All experiments used exactly the same sequence, obtained by seeding the random() function.

Table 26 describes the parameters used for each experiment, other than the number of passes which was held constant at five in all cases. Columns in this table are interpreted as follows:

- 'Exp.' is the experiment number.
- 'Pop.' is the population size.
- The 'Max. Gen.' column contains two values for each experiment. The first value is the maximum number of generations per pass which will be permitted under any circumstances. The second value is the *stagnation limit*, *i.e.* the

maximum permitted number of generations in which no improvement occurs (see Section 4.4.3.3.1).

- 'Fitness' describes the fitness function used. These functions are described in greater detail in Section 4.4.3.5.1. The term 'adjusted' indicates that both the penalty and the bonus described in Section 4.4.3.5.2 were applied.

- 'Overpop.' indicates which form of overpopulation is used; the term 'original' denotes the version described in [McCallum90], while the term 'modified' denotes the version described in Section 4.4.3.3.2.

- 'Missing' describes the manner in which missing or unknown attribute values in instances are handled. Here, 'always' indicates that an attribute with a missing value is always deemed to satisfy the requirements for that attribute in any rule set; 'never' means that an attribute with a missing value always fails to satisfy the requirements for that attribute in any rule set; 'discard' means that all training instances containing unknown values will be ignored entirely, and 'extra' indicates that unknown values will be handled by considering **unknown** to be yet another potential value for the given attribute.

- 'Con.' indicates whether or not convergence is evaluated as a potential termination condition, as described in Section 4.4.3.3.1.

- 'Operators' lists the operators which were activated during each experiment. See Table 24 for a list of operator names, and Section 4.4.3.2 for a description of each operator.

Exp.	Pop.	Max. Gen.	Fitness	Overpop.	Missing	Con.	Operators
1	100	600 / 300	p^2 adjusted	modified	always	yes	cross2, mutation
2	100	600 / 300	p^2 adjusted	modified	always	no	cross2, mutation
3	1000	600 / 300	p^2 adjusted	modified	always	no	cross2, mutation
4	100	1200 / 1200	p^2 adjusted	modified	always	no	cross2, mutation
5	100	600 / 300	p^2 adjusted	original	always	no	cross2, mutation
6	100	600 / 300	p^2	modified	always	no	cross2, mutation
7	100	600 / 300	p	modified	always	no	cross2, mutation
8	100	600 / 300	$10 \cdot p$	modified	always	no	cross2, mutation
9	100	600 / 300	p^3	modified	always	no	cross2, mutation
10	100	600 / 300	e^p	modified	always	no	cross2, mutation
11	100	600 / 300	$1 - \mathcal{L}(n, e)$	modified	always	no	cross2, mutation
12	100	600 / 300	$comp \cdot (1 - cons)$	modified	always	no	cross2, mutation
13	100	600 / 300	p^2 adjusted	modified	never	no	cross2, mutation
14	100	600 / 300	p^2 adjusted	modified	discard	no	cross2, mutation
15	100	600 / 300	p^2 adjusted	modified	extra	no	cross2, mutation
16	100	600 / 300	p^2 adjusted	modified	always	no	cross1, mutation
17	100	600 / 300	p^2 adjusted	modified	always	no	crossU, mutation
18	100	600 / 300	p^2 adjusted	modified	always	no	crossMU, mutation
19	100	600 / 300	p^2 adjusted	modified	always	no	cross2, addalt, dropcond, genspec, mutation
20	100	600 / 300	p^2 adjusted	original	always	yes	cross2, addalt, dropcond, genspec, mutation

Table 26: ELGAR experiment descriptions

The intentions behind these experiments were as follows:

- Experiment 1 is intended to establish the behaviour of ELGAR's genetic mode with the most commonly used parameters.
- Experiment 2 demonstrates the effect of the convergence test as a termination criterion.
- Experiments 3 and 4, respectively, test the effects of a larger than usual population and a greater than usual limit on the number of generations evaluated.
- Experiment 5 compares the effects of the original and modified forms of overpopulation.
- Experiments 6 through 12 demonstrate the effects of different fitness functions.

- Experiments 13 through 15 demonstrate the effects of different methods of handling unknown attribute values.
- Experiments 16 through 19 show the effects of different operators.
- Experiment 20 evaluates the effect of the convergence test when a greater number of operators are in use.

The results of the twenty experiments are summarized in Tables 27 - 34. Columns in these tables are interpreted as follows:

- 'Exp.' is the experiment number.
- 'Fit.' is the average over all passes of the fitness of the best observed individual from each pass. Note that, in the experiments described here, every run included five passes.
- 'CPU' is the average CPU time consumed per pass, in seconds.
- 'Gen.' is the average number of generations per pass.
- 'Avg. Train' is the average over all passes of the classification accuracy of the best observed individual from each pass, measured against the training set. This is shown both as a percentage of all training instances, and (under the subheading '#') as the actual number of training instances predicted correctly, as compared to the number of instances in the entire training set.
- 'Max Train' is the classification accuracy of the best individual observed during the entire run, as measured against the training set. This is shown both as a percentage and as the actual number of training instances predicted correctly.
- The 'Avg. Recall' and 'Max Recall' columns refer to the same individuals as those described by 'Avg. Train' and 'Max Train', and provide similar information. However, it is important to note that the values in these two columns do not necessarily reflect the best observed performance relative to the recall set; instead they describe the performance against the recall set of those individuals which performed best against the training set — a distinction which is subtle, but important. In order for the learning process to be fair, the recall set was not made available to the learner until after the training phase was complete.

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	1.607	15.8	74	78.5	150/191	82.7	158	70.3	67/95	76.8	73
2	2.411	138.8	568	85.4	163/191	87.4	167	70.7	67/95	75.8	72
3	2.902	1336.9	505	89.6	171/191	91.1	174	69.7	66/95	73.7	70
4	2.083	238.9	1200	82.6	158/191	85.3	163	70.3	67/95	73.7	70
5	2.047	149.8	514	82.3	157/191	85.9	164	74.1	70/95	77.9	74
6	0.729	147.6	542	85.3	163/191	88.0	168	69.1	66/95	75.8	72
7	0.853	147.1	542	85.3	163/191	88.0	168	69.1	66/95	75.8	72
8	8.534	147.3	542	85.3	163/191	88.0	168	69.1	66/95	75.8	72
9	0.622	147.3	542	85.3	163/191	88.0	168	69.1	66/95	75.8	72
10	2.348	147.4	542	85.3	163/191	88.0	168	69.1	66/95	75.8	72
11	0.984	131.4	552	64.3	123/191	77.5	148	41.3	39/95	45.3	43
12	0.799	147.5	591	83.8	160/191	88.0	168	72.2	69/95	76.8	73
13	2.205	104.3	541	83.7	160/191	86.9	166	67.4	64/95	72.6	69
14	2.202	100.8	541	84.3	157/186	87.6	163	69.2	63/91	74.7	68
15	2.228	110.5	510	83.9	160/191	86.4	165	72.6	69/95	76.8	73
16	2.120	111.5	514	82.9	158/191	87.4	167	70.3	67/95	76.8	73
17	2.172	110.5	498	83.4	159/191	91.1	174	69.1	66/95	73.7	70
18	2.033	115.7	553	82.2	157/191	87.4	167	69.5	66/95	74.7	71
19	2.605	401.9	600	87.1	166/191	89.0	170	71.2	68/95	73.7	70
20	2.594	6382.1	591	87.2	167/191	90.1	172	65.7	62/95	70.5	67

Table 27: ELGAR experimental results for the Ljubljana Breast Cancer Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	4.808	6.3	40	90.0	112/124	100.0	124	80.1	346/432	99.1	428
2	5.784	23.2	165	98.2	122/124	100.0	124	95.7	414/432	100.0	432
3	5.994	65.3	49	100.0	124/124	100.0	124	100.0	432/432	100.0	432
4	5.995	42.2	310	100.0	124/124	100.0	124	98.5	426/432	100.0	432
5	5.784	26.5	156	98.2	122/124	100.0	124	96.4	416/432	100.0	432
6	0.966	16.2	124	98.2	122/124	100.0	124	96.3	416/432	100.0	432
7	0.982	16.2	124	98.2	122/124	100.0	124	96.3	416/432	100.0	432
8	9.823	16.2	124	98.2	122/124	100.0	124	96.3	416/432	100.0	432
9	0.951	16.3	124	98.2	122/124	100.0	124	96.3	416/432	100.0	432
10	2.672	16.2	124	98.2	122/124	100.0	124	96.3	416/432	100.0	432
11	0.982	34.4	287	94.2	117/124	100.0	124	93.6	404/432	100.0	432
12	0.970	33.7	226	98.2	122/124	100.0	124	95.4	412/432	100.0	432
13	5.784	23.1	165	98.2	122/124	100.0	124	95.7	414/432	100.0	432
14	5.784	23.0	165	98.2	122/124	100.0	124	95.7	414/432	100.0	432
15	5.400	44.8	370	95.0	118/124	100.0	124	89.3	386/432	100.0	432
16	5.573	22.0	204	96.5	120/124	100.0	124	93.1	402/432	100.0	432
17	5.781	23.0	164	98.2	122/124	100.0	124	95.6	413/432	100.0	432
18	5.362	46.1	410	94.7	117/124	100.0	124	89.2	385/432	100.0	432
19	5.981	33.3	94	100.0	124/124	100.0	124	89.8	388/432	93.5	404
20	4.789	502.2	403	89.8	111/124	91.1	113	81.7	353/432	83.3	360

Table 28: ELGAR experimental results for the MONK's Problem 1 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	2.492	13.4	72	80.6	136/169	84.6	143	72.1	311/432	73.6	318
2	2.616	85.2	446	81.7	138/169	88.8	150	73.4	317/432	77.8	336
3	3.250	896.8	393	87.1	147/169	89.3	151	77.0	333/432	81.5	352
4	2.616	217.7	1200	81.7	138/169	85.2	144	71.0	307/432	75.5	326
5	2.823	126.2	408	83.4	141/169	87.0	147	70.7	306/432	73.6	318
6	0.717	108.6	496	84.6	143/169	88.8	150	73.9	319/432	77.8	336
7	0.846	108.5	496	84.6	143/169	88.8	150	73.9	319/432	77.8	336
8	8.462	108.7	496	84.6	143/169	88.8	150	73.9	319/432	77.8	336
9	0.608	108.9	496	84.6	143/169	88.8	150	73.9	319/432	77.8	336
10	2.331	108.8	496	84.6	143/169	88.8	150	73.9	319/432	77.8	336
11	0.969	97.0	482	80.8	137/169	88.2	149	74.8	323/432	81.5	352
12	0.692	156.5	547	83.7	141/169	92.9	157	71.7	310/432	78.7	340
13	2.616	84.3	446	81.7	138/169	88.8	150	73.4	317/432	77.8	336
14	2.616	83.9	446	81.7	138/169	88.8	150	73.4	317/432	77.8	336
15	2.411	73.9	420	79.9	135/169	84.6	143	70.9	306/432	71.3	308
16	2.288	59.3	362	78.8	133/169	85.8	145	67.5	292/432	71.3	308
17	2.507	74.7	394	80.7	136/169	89.3	151	71.2	307/432	76.6	331
18	2.645	86.6	431	81.9	138/169	91.1	154	72.0	311/432	76.4	330
19	4.643	795.2	411	99.2	168/169	100.0	169	79.0	341/432	81.5	352
20	4.283	1127.8	581	96.1	162/169	98.8	167	78.0	337/432	84.5	365

Table 29: ELGAR experimental results for the MONK's Problem 2 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	5.190	2.6	20	93.9	115/122	94.3	115	97.0	419/432	97.2	420
2	5.309	27.7	352	94.9	116/122	95.9	117	97.3	420/432	97.5	421
3	5.464	365.1	352	96.2	117/122	97.5	119	95.3	412/432	96.3	416
4	5.425	116.8	1200	95.9	117/122	95.9	117	96.5	417/432	97.5	421
5	5.425	52.1	348	95.9	117/122	95.9	117	95.9	414/432	96.3	416
6	0.920	37.3	357	95.9	117/122	95.9	117	96.6	417/432	97.7	422
7	0.959	37.3	357	95.9	117/122	95.9	117	96.6	417/432	97.7	422
8	9.590	37.4	357	95.9	117/122	95.9	117	96.6	417/432	97.7	422
9	0.882	37.3	357	95.9	117/122	95.9	117	96.6	417/432	97.7	422
10	2.609	37.4	357	95.9	117/122	95.9	117	96.6	417/432	97.7	422
11	0.982	71.4	518	94.9	116/122	98.4	120	88.3	382/432	94.0	406
12	0.923	44.4	340	95.9	117/122	95.9	117	96.8	418/432	98.1	424
13	5.309	27.4	352	94.9	116/122	95.9	117	97.3	420/432	97.5	421
14	5.309	27.3	352	94.9	116/122	95.9	117	97.3	420/432	97.5	421
15	5.425	45.8	443	95.9	117/122	95.9	117	96.7	418/432	97.2	420
16	5.425	33.6	374	95.9	117/122	95.9	117	96.2	416/432	97.2	420
17	5.346	46.3	390	95.2	116/122	95.9	117	97.4	421/432	100.0	432
18	5.309	26.2	351	94.9	116/122	95.9	117	96.9	418/432	97.2	420
19	5.824	185.0	348	99.3	121/122	100.0	122	88.2	381/432	91.0	393
20	5.519	1409.4	464	96.7	118/122	98.4	120	95.0	410/432	100.0	432

Table 30: ELGAR experimental results for the MONK's Problem 3 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	0.226	5.7	1	47.7	1937/4062	47.7	1937	48.7	1979	48.7	1979
2	2.432	1184.4	414	68.5	2783/4062	99.8	4053	69.1	2808	99.8	4052

Table 31: ELGAR experimental results for the Mushrooms Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	1.176	51.4	45	60.8	762/1252	66.5	832	59.5	745/1252	64.2	804
2	1.970	626.7	487	67.9	850/1252	69.2	867	63.6	797/1252	64.8	811

Table 32: ELGAR experimental results for the Ludwig Trial V 17-Attribute Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	0.278	1.7	1	52.8	661/1252	52.8	661	53.6	671/1252	53.6	671
2	0.278	167.4	301	52.8	661/1252	52.8	661	53.6	671/1252	53.6	671

Table 33: ELGAR experimental results for the Ludwig Trial V 31-Attribute Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
1	3.923	14.4	38	95.1	333/350	96.6	338	91.5	319/349	95.7	334
2	4.455	207.5	575	99.5	348/350	100.0	350	94.6	330/349	95.4	333
3	4.503	1136.6	302	99.9	350/350	100.0	350	94.7	330/349	96.6	337
4	4.484	358.6	1115	99.8	349/350	100.0	350	94.6	330/349	95.4	333
5	4.449	275.8	523	99.5	348/350	100.0	350	93.7	327/349	94.0	328
6	0.995	175.1	478	99.8	349/350	100.0	350	94.4	329/349	96.8	338
7	0.998	174.8	478	99.8	349/350	100.0	350	94.4	329/349	96.8	338
8	9.977	174.9	478	99.8	349/350	100.0	350	94.4	329/349	96.8	338
9	0.993	174.7	478	99.8	349/350	100.0	350	94.4	329/349	96.8	338
10	2.712	175.1	478	99.8	349/350	100.0	350	94.4	329/349	96.8	338
11	0.996	214.8	528	99.3	348/350	99.7	349	93.6	327/349	95.1	332
12	0.992	248.5	600	99.5	348/350	99.7	349	94.2	329/349	95.7	334
13	4.326	189.4	574	98.5	345/350	98.6	345	92.4	323/349	93.1	325
14	4.448	221.8	595	99.3	343/345	99.7	344	93.0	314/338	95.9	324
15	4.387	221.6	570	99.0	346/350	99.1	347	94.1	328/349	94.8	331
16	4.436	190.2	552	99.4	348/350	100.0	350	93.4	326/349	94.6	330
17	4.407	195.2	506	99.1	347/350	99.7	349	93.9	328/349	96.3	336
18	4.380	172.3	541	98.9	346/350	99.4	348	93.6	327/349	95.1	332
19	4.488	118.2	71	100.0	350/350	100.0	350	91.5	319/349	93.1	325
20	4.487	397.3	46	100.0	350/350	100.0	350	92.2	322/349	93.7	327

Table 34: ELGAR experimental results for the Wisconsin Breast Cancer Data Set

Table 35 lists the best result obtained for each data set using ELGAR, along with results for C4 (as determined directly by experiment). For each ELGAR result, the number of the experiment or experiments in which that result was obtained is shown as well. The 'Baseline' column indicates the baseline accuracy for the recall set in each case.

Data Set	Baseline	Results		Exp.
		C4	ELGAR	
Ljubljana	73.7%	76.8%	77.9%	5
MONK's 1	50.0%	75.9%	100.0%	2 - 18
MONK's 2	67.1%	65.1%	84.5%	20
MONK's 3	52.8%	97.2%	100.0%	17 - 20
Mushrooms	51.3%	100.0%	99.8%	2
Trial V/17	53.6%	65.0%	64.8%	2
Trial V/31	53.6%	61.7%	53.6%	1, 2
Wisconsin	66.2%	93.7%	96.8%	6 - 10

Table 35: ELGAR compared with C4 on recall sets

4.6 Conclusions from Experiments with ELGAR

ELGAR's results suggest a number of conclusions, chief among which are:

- The genetic algorithm is surprisingly resistant to perturbation by means of altered operating parameters. The twenty experiments performed here involve a number of widely different conditions, but the results of those experiments are remarkably similar, which suggests that the potential benefit to be gained from minor changes to the model is limited; it would appear that significant improvement would require a more fundamental change, such as a different data representation, or a combination of genetic search with other learning methods.

- Although the difference between experimental outcomes is small, it is not nonexistent; however, there is no single experimental design (*i.e.* no single set of operating parameter values) which is consistently superior for all data sets.

  The lack of any case in which experiment 3 was superior suggests that enlarging the population is not necessarily useful; likewise, the lack of any case in which

experiment 4 was superior suggests that increasing the maximum generation limit is unlikely to help.

However, other techniques which increase exploration at the expense of CPU time, such as those employed in experiments 2 (not checking for convergence) and 20 (using the original McCallum and Spackman version of overpopulation [McCallum90] in conjunction with a greater number of operators) appear to be fruitful in some learning domains.

- Due to the use of tournament selection, the specific choice of fitness function appears to play a relatively small role in ELGAR. Different functions of the percentage of correctly classified training instances, as exemplified by experiments 6 - 10, appear to have no influence whatsoever. Laplace error and completeness vs. consistency, as tested in experiments 11 and 12, do exhibit slightly different behaviour, but not enough of a difference to prevent being overshadowed by the effects of other parameters.

- Adjustments made to the fitness function, as described in Section 4.4.3.5.2, do appear to make a difference for the better. Evidence for this conjecture arises from the fact that experiments 2 and 6 are identical except for these fitness adjustments, and experiment 2 consistently outperforms experiment 6, except in the atypical results obtained for the Wisconsin breast cancer data set; here, the unadjusted fitness function clearly provides better performance, although it is not clear why this should be so.

- It is interesting to compare ELGAR's genetic search to that of GABIL. This cannot be done directly, since the only results available for GABIL are for the Ljubljana data set and were obtained using the entire data set for both training and recall [De Jong93, p. 168]. However, what can be compared is the degree of improvement with respect to the baseline accuracy of the recall set used in each case.

For GABIL, the baseline accuracy of the entire data set is 70.3%, and the best reported result is 72.0% (see Section 4.4.2.6), which represents an absolute improvement of

$$72.0\% - 70.3\% = 1.7\%$$

and a relative improvement of

$$(72.0\% - 70.3\%) \div 70.3\% = 2.4\%$$

For ELGAR, the baseline accuracy of the recall set is 73.7%, and the best result obtained was 77.9% (see Table 35). This represents an absolute improvement of

$$77.9\% - 73.7\% = 4.2\%$$

and a relative improvement of

$$(77.9\% - 73.7\%) \div 73.7\% = 5.7\%$$

This is certainly not conclusive, but it does suggest that overpopulation might be a more successful technique than the within-problem self-adaptation exhibited by GABIL.

- ELGAR is generally competitive with C4, outperforming it in five of eight data sets. ELGAR does particularly well compared to C4 on the MONK's problems, but does particularly poorly on both versions of the Ludwig Trial V data set.

- Other than the fitness function, the single item which most influences genetic search is the data representation format. This would appear to be an interesting area for further study, particularly because the format used in GABIL and inherited by ELGAR appears vulnerable to **epistasis**, the condition in which bits that are closely related (in the sense that they are part of the same building block) are widely separated physically within a bit string; this condition causes building blocks to have long defining lengths, making them vulnerable to disruption. By allowing attributes to occur in arbitrary order in the bit string for each rule set, GABIL and ELGAR run the risk that some attributes may be closely related and yet be located in widely separated areas of the bit string.

  For example, consider a modified version of the lion example from Section 1.1.3.1, as shown in Table 36.

Furry?	Age	Tail	Name	Size	Class
yes	old	long	Leo	large	lion
yes	young	short	Jane	medium	lion
yes	young	long	Spot	large	lion
no	young	medium	Fido	large	not lion
yes	old	long	Rover	small	not lion
yes	young	short	Spike	small	not lion
no	young	tiny	Pinky	small	not lion
no	old	huge	TRex	large	not lion

Table 36: A modified training set for the concept LION

The Tail and **Name** attributes are clearly not relevant, and have been added only to illustrate a point. The number of bits required to represent a rule would change from seven for the original lion example (two bits for **Furry?**, plus two for **Age** and three for **Size**) to a total of twenty (the original seven, plus five for Tail and eight for **Name**). An example of a concept description for this modified training set might be

(Furry = yes) ∧ (Size = medium ∨ large) → lion

or

10 11 11111 11111111 011

Suppose that this is actually the correct description for the LION concept, and that the most accurate rule set currently in the population is

10 11 11011 11110111 011

Consider what will happen when one-point or two-point crossover is applied to this individual. Almost any possible choice of crossover point or points would cause the current values of the **Furry?** and **Size** features to be lost.

This problem occurs because the bits which are actually part of the solution (in this case bit positions 0, 1, 17, 18 and 19) are separated by a number of bit positions which describe irrelevant features (in this case the gap between bits 1 and 17). In the general case, this effect will occur in any situation in which bit positions that are related in terms of the solution are far apart from each other.

In contrast, suppose the attributes were reordered as shown in Table 37.

Furry?	Size	Age	Tail	Name	Class
yes	large	old	long	Leo	lion
yes	medium	young	short	Jane	lion
yes	large	young	long	Spot	lion
no	large	young	medium	Fido	not lion
yes	small	old	long	Rover	not lion
yes	small	young	short	Spike	not lion
no	small	young	tiny	Pinky	not lion
no	large	old	huge	TRex	not lion

Table 37: A modified and reordered training set for the concept LION

In this representation, the same rule

(Furry = yes) ∧ (Size = medium ∨ large) → lion

is now encoded as

```
10 011 11 11111 11111111
```

Here the relevant bit positions are all adjacent. The difference between this situation and the previous one is best illustrated by the following pair of schemata:

```
10 ** ***** ******** 011
10 011 ** ***** ********
```

Clearly the second schema offers far more possibilities for crossover cut points that would not cause disruption.

One possible way of dealing with this problem in general is to reorder attributes according to some measure of relevance, in the hope that this will bring relevant bits together. See Section 5.1.3.2 for an elaboration of this point. Another alternative would be to attempt to create a completely different data representation in which the order of attributes would not be an issue; however, the problem with this idea is that it is not at all clear what such a representation format might be.

Chapter 5

Hybrid ELGAR

Given that there is room for ELGAR's performance to improve, one potential means of doing so is to combine genetic search with one or more other techniques. Some approaches already taken by other researchers include:

- using genetic search to select attributes to be used by a different learning algorithm (*e.g.* [Vafaie92, Vafaie94])
- using genetic search to determine neural network topology (*e.g.* [Opitz97])
- integration of genetic learning with Michalski's AQ algorithm (*e.g.* [Bala94])

A simpler approach is to combine genetic learning with ideas inspired by Holte's 1R classifier [Holte93]; this is the approach discussed in Section 5.1.

5.1 Integration of ELGAR and 1R

Holte's recommended "simplicity first" approach to learning (see Section 3.2.4) would create a concept description by beginning with the simplest description possible, and adding whatever complexity might be required to capture knowledge that would otherwise be missed. In contrast, genetic learning searches through the entire space of possible concept descriptions which can be expressed using the available data representation format. These two approaches are quite dissimilar, for which reason a means of combining them is not obvious; however, their very dissimilarity would appear to make them good candidates to work together.

5.1.1 Attribute Subsets

A natural approach to the integration of "simplicity first" into genetic learning is to confine learning to a single attribute (which 1R already does, of course), then expand by considering subsets of two attributes, then three, adding an attribute at a time until an appropriate level of accuracy is reached. This idea is a natural outgrowth of the 2R extension of 1R described in [Nevill-Manning95].

Although this approach is tempting, it is impractical because it requires too much computation. For example, the Ludwig Trial V 31-attribute data set contains 31 attributes; even if the attribute subset is limited to a cardinality of ten, this still requires the investigation of a total of

$$\sum_{i=1}^{10} {}_{31}C_i = 75,973,188$$

subsets, each one subject to its own genetic search. Considering that results should ideally be available within the experimenter's lifetime, this might not be a useful strategy.

5.1.1.1 Bounded Attribute Subsets

Given that exploration of all possible attribute subsets is not practical, the next approach is to require that every subset under investigation must contain the single attribute which leads to the best available one-rule for the given data set. This does indeed reduce the number of subsets to be evaluated, but not by enough of a margin; for example, in the Ludwig Trial V 31-attribute data set, there would still be a total of

$$31 + \sum_{i=1}^{9} {}_{30}C_i = 22,964,117$$

attribute subsets of cardinality ten or less to consider. This number is derived from the fact that the individual attributes must still be evaluated to determine which one is best (the initial term of the sum), and then the remaining zero to nine elements of each set must be selected from the remaining attributes. This represents a significant reduction with respect to the original number of subsets, but clearly not to a sufficient extent.

5.1.1.2 Incremental Attribute Subsets

The next approach is to construct subsets one attribute at a time, adding only the attribute which contributes the greatest gain. Informally this can be stated as follows; a more formal statement of this algorithm is shown in Figure 13.

- Begin by evaluating one-rules involving each attribute, and determine which attribute leads to the best possible rule; call this attribute `F1`.
- Evaluate two-rules (rules containing exactly two attributes) involving `F1` paired with every remaining attribute; let `F2` be the attribute which, with `F1`, gives the best possible rule set.
- Evaluate three-rules involving `F1` and `F2`, letting `F3` be the attribute which completes the best-performing triple.
- Continue as above until all attributes have been evaluated or until the desired level of accuracy has been reached.

This version reduces the number of subsets which must be evaluated to

$$\sum_{i=1}^{N} i = \frac{N \cdot (N+1)}{2}$$

For ten attributes, this reduces to

$$\frac{10 \cdot 11}{2} = 55$$

(specifically, ten subsets of cardinality 1, nine subsets of cardinality 2, down to only one subset of cardinality 10). This number is low enough to be achievable, but even so in practice this technique requires much more CPU time than a pure genetic search (for example, one run using the 17-attribute version of the Ludwig Trial V data set ran for a total of 37,278.8 seconds); unfortunately, it does not yield a comparable increase in accuracy. While it may be simple conceptually, it proves to be far from simple computationally.

1. Initialize **attribute_count** to zero, and **best_attribute_subset** to the empty set.

2. Increment **attribute_count**.

3. Set **attribute_subset** to the current value of **best_attribute_subset**.

4. For every attribute **A** which is not currently a member of **attribute_subset**:

   (a) Add **A** to **attribute_subset**.

   (b) Evaluate **attribute_subset**, as follows:

      i. Compute **N** = number of bits required to represent a single rule using the attributes in **attribute_subset**.

      ii. If **N** < threshold then
          Exhaustively build and test all possible rules
          using the attributes in **attribute_subset**.
      else
          Use genetic search to find the best possible rules
          using the attributes in **attribute_subset**.

   (c) If the rules developed for this **attribute_subset** perform better than those for the current **best_attribute_subset**, then replace **best_attribute_subset** by **attribute_subset**.

   (d) Remove **A** from **attribute_subset**.

5. Repeat steps 2 - 4 until (**attribute_count** = total attributes) or 100% training accuracy is achieved.

Figure 13: Incremental algorithm for constructing and evaluating subsets

Another drawback of this method is that it risks overlooking combinations of attributes which together perform better than either one alone. For example, suppose that attributes **A1**, **A2** and **A3** in some arbitrary data set produce one-rules with accuracies of 70%, 20% and 30% respectively; despite the relatively poor accuracy obtained using either **A2** or **A3** alone, in theory it could happen that the combination of **A2** and **A3** together might yield a rule set which would be more accurate than any involving **A1**. For these two reasons, this line of investigation was abandoned.

5.1.2 One-Rules as Seeds

There is nothing in the definition of the classical genetic algorithm which requires the initial population to contain only randomly generated individuals; instead, it is perfectly acceptable to seed the initial population with known candidate solutions, just as a lake might be stocked to guarantee a supply of fish. Of course, most of the time no good candidates will be known *a priori*. This train of thought suggests the simplest possible integration of 1R with genetic learning, and the one which ultimately proved to be most fruitful: in particular, evaluate all possible one-rules, and add the best one-rule for each attribute to the initial population for the genetic search.

In Hybrid ELGAR, each one-rule is represented in its natural form, which is a rule in which all bits are set to one other than those which correspond to the attribute for which the one-rule is defined. For example, using the WIDGET concept and attributes from Table 21, suppose that the best one-rule for the **colour** attribute is

```
(colour = red ∨ green) → widget
```

This rule would be added to Hybrid ELGAR's initial population, represented as:

```
111 10010 11
```

Thus the **size** and **shape** attributes are not used in this rule . Variations which were tested and which proved not to be useful include:

- changing approximately half of the bits for unused attributes to zero (*e.g.* a representation of `011 10010 01` for the above rule)
- changing approximately one third of the bits for unused attributes to zero (*e.g.* a representation of `110 10010 11` for the above rule)

- keeping all bits for all unused attributes set to one, but also adding a second copy of each rule with approximately one third of the unused attribute bits set to zero (*e.g.* a representation of `111 10010 11 \bullet 110 10010 11` for the above rule)

- seed rule sets composed of the one-rule for a given attribute, plus a random number of randomly chosen one-rules involving other attributes, with an enforced lack of duplication among the rules in a given rule set; for example, given the above rule, suppose that the best one-rule for **size** is

  `(size = medium` $\vee$ `large)` $\rightarrow$ `widget`

  or

  `011 11111 11`

  and that the best one-rule for **shape** is

  `(shape = circle)` $\rightarrow$ `widget`

  or

  `111 11111 10`

  Then a sample seed individual for the **colour** attribute might look like

  `111 10010 11`
  `011 11111 11`

  or

  `(colour = red` $\vee$ `green)` $\rightarrow$ `widget`
  `(size = medium` $\vee$ `large)` $\rightarrow$ `widget`

- seed rules in which most unused attributes are represented by all one bits, but a random number of randomly chosen attributes are represented by the bit string for that attribute, taken from the best one-rule involving that attribute; for example, if the best one-rule for **size** is

  `(size = small` $\vee$ `large)` $\rightarrow$ `widget`

then the one-rule for the **colour** attribute might include that rule, becoming

```
101 10010 11
```

Since the population size (100 unless otherwise specified by the user) is usually much larger than the number of attributes in most data sets, and given that only one rule will be seeded per attribute, most of the time the initial population will still include a number of randomly generated rule sets. However, those will almost always be much less fit than the best available one-rules, causing the genetic search to be biased in favour of the portion of the search space which contains those one-rules. Thus, this method is reasonably close to the spirit of Holte's concept of "simplicity first".

5.1.3 One-Rules and Attribute Ranking

The one-rules generated by 1R implicitly define a ranking of the available attributes, in descending order of classification accuracy on the training set. This information can be used in at least two different ways.

5.1.3.1 Attribute Filtering

The topic of attribute selection was touched upon in Section 1.1.3.1, in which it was mentioned that the correct choice of attributes can influence the quality of learning in a given domain. The term *attribute filtering* refers to removing attributes from a domain once they are determined not to be useful, a practice which, if successful, allows the learning process to concentrate on the presumably relevant attributes that remain.

The ranking defined by 1R can be used for attribute filtering, as originally proposed in [Holmes95]. As performed in Hybrid ELGAR, attribute filtering specifically refers to removing from the data set any attribute for which the best available one-rule has a classification accuracy on the training set which is less than the training set's baseline accuracy. Note that if the best one-rules for all attributes are below baseline, none will be removed; this fairly obvious special case would otherwise make learning impossible.

5.1.3.2 Attribute Reordering

As discussed in Section 4.2.6.2, the success of genetic search depends directly on the quality and defining length of the best available building blocks; the longer the defining length of a building block, the less likely it is to be created by recombination, and the more likely it is to be broken up. This has a direct consequence on the design of the data representation for genetic learning, in the sense that bits in the bit string which are likely to be related to each other should be located as close to each other as possible.

Until a data representation scheme significantly better than that of GABIL is designed, Pittsburgh-style genetic learners are limited in their ability to adhere to this principle, since their only degree of freedom is the ordering of attributes within the rule (see Section 4.6 for an example). While the problem of finding the optimal attribute order is as yet unsolved, one obvious candidate is to order attributes based on their 1R ranking, which is at least an approximation of their relevance. This does not guarantee anything, but it would appear to increase the likelihood that building block defining length will be minimized, to an extent proportional to the correlation between 1R ranking and the actual relevance of each attribute. Thus, Hybrid ELGAR allows the user to choose whether to sort all attributes into descending order of 1R ranking.

5.1.4 One-Rules as a Partition of the Training Set

Like any rule, the best one-rule returned by 1R partitions the training set into those instances which it classifies correctly, and those which it does not. Typically the former subset is much larger than the latter, which in light of [Holte93] should probably not be surprising. Thus, the task of learning how to classify the comparatively few training instances which the best one-rule misclassifies is likely to be much easier than learning to classify the entire training set. This suggests the approach shown in Figure 14.

1. Build all one-rules, and find the one-rule with the highest classification accuracy over the entire training set. If this rule classifies the entire training set correctly, stop.

2. Use the best one-rule from step 1 to classify all training instances. Mark as solved all training instances which are classified correctly.

3. Train the genetic learner on only those training instances which are not marked solved.

Figure 14: Learning in a partitioned training set

This procedure will end with a partition of the training set (*i.e.* two disjoint subsets whose union is the original set), such that one subset contains instances which are all classified correctly by the best one-rule, and the other contains instances of which all (or at least many) are classified correctly by a genetically learned rule set.

Of course, this leaves open the question of how to deal with recall instances; when a previously unseen instance is encountered, into which subset should it be added? Clearly what is needed at this point is a meta-rule set which would distinguish between the two subsets — *i.e.* a rule set with which to decide whether a given recall instance should be classified by the best one-rule found, or by the genetically learned rule set.

The term *meta-rule* is used in order to make it clear that the purpose of these rules is to decide which ordinary rules should be applied to the classification of each recall instance. The format of a meta-rule is no different than the format of any other rule, in that a meta-rule includes conditions based on attribute values, and an outcome which classifies an instance; the difference between a meta-rule and an ordinary rule is the type of classification produced, since a meta-rule classifies an instance according to which rule or rules to apply, while ordinary rules classify an instance as a member or non-member of the concept to be learned.

Thus, the next step is to begin a new genetic search, over all training instances, in order to find the appropriate meta-rules. In some sense, one learning problem has been replaced by a new one, which may be either easier or more difficult to solve. Experimental results (see Section 5.2) suggest that, in most domains, the meta-rule search and the original learning problem appear to be of an equivalent level of difficulty.

The meta-rule approach complicates the problem of evaluating fitness; in particular, under what circumstances is an instance considered to be covered by a rule set? As implemented in Hybrid ELGAR, this determination is made as follows:

1. If the instance is covered by the meta-rules, it belongs to the subset which is evaluated by the best one-rule. Thus, if it is covered by both the meta-rules and the best one-rule, then it is considered covered overall.

2. Otherwise, if the instance is not covered by the meta-rules, but is covered by the rule set from the original genetic search, then it is considered covered overall.

3. Otherwise the instance is considered not to be covered overall.

5.2 Experimental Results

Experiments with Hybrid ELGAR were done on the same machine as those done using ELGAR (Section 4.5.2). Table 38 describes the parameters used for each experiment, other than the following which were constant in all cases:

- A total of five passes were performed.
- The population size was 100.
- The maximum number of generations was 600.
- The maximum number of generations without improvement (*stagnation limit*) was 300.
- The fitness function was p^2, adjusted by applying both the penalty and the bonus described in Section 4.4.3.5.2 (except in experiment 22, the only one in which the fitness penalty and bonus were **not** applied).
- Unknown attribute values in instances were always deemed to satisfy the requirements for that attribute in any rule set.

Exp.	Overpop.	Convergence	Filtering	Sorting	Meta-Rules	Operators
21	modified	no	no	no	no	cross2, mutation
22	modified	no	no	no	no	cross2, mutation
23	modified	no	no	yes	no	cross2, mutation
24	modified	no	yes	no	no	cross2, mutation
25	modified	no	yes	yes	no	cross2, mutation
26	modified	yes	yes	no	no	cross2, mutation
27	modified	yes	yes	yes	no	cross2, mutation
28	original	no	yes	no	no	cross2, addalt, dropcond, genspec, mutation
29	original	yes	yes	no	no	cross2, addalt, dropcond, genspec, mutation
30	modified	no	yes	no	yes	cross2, mutation

Table 38: Hybrid ELGAR experiment descriptions

Columns in Table 38 are interpreted as follows:

- 'Exp.' is the experiment number, which begins at 21 because experiments 1 – 20 were performed with ELGAR, as described in Section 4.5.2.
- 'Overpop.' indicates which form of overpopulation is used; the term 'original' denotes the version described in [McCallum90], while the term 'modified' denotes the version described in Section 4.4.3.3.2.
- 'Convergence' indicates whether or not convergence is evaluated as a potential termination condition, as described in Section 4.4.3.3.1.
- 'Filtering' indicates whether or not attribute filtering was performed, as described in Section 5.1.3.1.
- 'Sorting' indicates whether or not attribute reordering was performed, as described in Section 5.1.3.2.
- 'Meta-Rules' indicates whether or not the search was performed in *meta-rule mode*, as described in Section 5.1.4.
- 'Operators' lists the operators which were activated during each experiment.

The intentions behind these experiments were as follows:

- Experiment 21 is intended to establish the behaviour of Hybrid ELGAR with none of the experimental features enabled.
- Experiment 22 is similar to experiment 21, except that the usual fitness penalty and bonus are not applied; this is the only experiment in this series for which that is true.
- Experiment 23 investigates the effect of attribute reordering without attribute filtering.
- Experiment 24 demonstrates the effect of attribute filtering without attribute reordering.
- Experiment 25 combines both attribute filtering and attribute reordering.
- Experiment 26 and 27 are similar to experiments 24 and 25, respectively, but with convergence testing enabled.
- Experiments 28 and 29 include the original McCallum and Spackman version of overpopulation [McCallum90] along with a greater number of operators; apart from this, they are similar to experiments 24 and 26 respectively.
- Experiment 30 explores the effect of partitioning the training set via *meta-rule mode*.

The results of the ten experiments are summarized in Tables 39 - 46. Columns in these tables are interpreted as follows:

- 'Exp.' is the experiment number.
- 'Fit.' is the average over all passes of the fitness of the best observed individual from each pass. Note that, in the experiments described here, every run included five passes.
- 'CPU' is the average CPU time consumed per pass, in seconds.
- 'Gen.' is the average number of generations per pass.

- 'Avg. Train' is the average over all passes of the classification accuracy of the best observed individual from each pass, measured against the training set. This is shown both as a percentage of all training instances, and (under the subheading '#') as the actual number of training instances predicted correctly, as compared to the number of instances in the entire training set.

- 'Max Train' is the classification accuracy of the best individual observed during the entire run, as measured against the training set. This is shown both as a percentage and as the actual number of training instances predicted correctly.

- The 'Avg. Recall' and 'Max Recall' columns refer to the same individuals as those described by 'Avg. Train' and 'Max Train', and provide similar information. However, it is important to note that the values in these two columns do not necessarily reflect the best observed performance relative to the recall set; instead they describe the performance against the recall set of those individuals which performed best against the training set — a distinction which is subtle, but important. In order for the learning process to be fair, the recall set was not made available to the learner until after the training phase was complete.

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	1.297	58.7	391	75.8	145/191	82.7	158	74.7	71/95	76.8	73
22	0.605	69.7	406	77.6	148/191	83.2	159	70.3	67/95	73.7	70
23	1.140	48.1	355	74.5	142/191	78.5	150	74.7	71/95	76.8	73
24	1.536	55.1	409	77.9	149/191	78.5	150	79.4	75/95	82.1	78
25	1.536	55.1	409	77.9	149/191	78.5	150	79.4	75/95	82.1	78
26	1.474	3.9	23	77.4	148/191	78.0	149	79.4	75/95	80.0	76
27	1.474	3.9	23	77.4	148/191	78.0	149	79.4	75/95	80.0	76
28	1.392	484.6	312	76.6	146/191	77.5	148	79.4	75/95	80.0	76
29	1.392	507.0	312	76.6	146/191	77.5	148	79.4	75/95	80.0	76
30	4.140	0.2	5	100.0	55/55	100.0	55	76.6	73/95	77.9	74

Table 39: Hybrid ELGAR experimental results for the Ljubljana Breast Cancer Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	5.114	33.6	331	92.6	115/124	100.0	124	88.3	382/432	100.0	432
22	0.907	29.0	248	95.2	118/124	100.0	124	90.5	391/432	100.0	432
23	5.668	33.5	291	97.3	121/124	100.0	124	94.4	408/432	100.0	432
24	5.114	33.6	331	92.6	115/124	100.0	124	88.3	382/432	100.0	432
25	5.668	33.6	291	97.3	121/124	100.0	124	94.4	408/432	100.0	432
26	4.428	5.8	57	86.8	108/124	91.1	113	76.4	330/432	83.3	360
27	4.222	7.5	92	85.0	105/124	92.7	115	73.9	319/432	83.3	360
28	4.637	505.2	419	88.5	110/124	91.1	113	80.0	346/432	83.3	360
29	4.637	508.2	419	88.5	110/124	91.1	113	80.0	346/432	83.3	360
30	5.999	0.1	2	100.0	33/33	100.0	33	89.8	388/432	89.8	388

Table 40: Hybrid ELGAR experimental results for the MONK's Problem 1 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	2.685	79.6	434	82.2	139/169	87.6	148	74.2	321/432	77.5	335
22	0.713	94.7	446	84.4	143/169	91.1	154	73.4	317/432	76.9	332
23	2.685	80.5	434	82.2	139/169	87.6	148	74.2	321/432	77.5	335
24	2.685	79.6	434	82.2	139/169	87.6	148	74.2	321/432	77.5	335
25	2.685	80.0	434	82.2	139/169	87.6	148	74.2	321/432	77.5	335
26	2.588	23.9	141	81.4	138/169	85.2	144	73.2	316/432	76.9	332
27	2.588	23.8	141	81.4	138/169	85.2	144	73.2	316/432	76.9	332
28	4.491	1727.9	586	97.9	165/169	98.8	167	78.1	338/432	79.4	343
29	4.491	1723.6	586	97.9	165/169	98.8	167	78.1	338/432	79.4	343
30	4.784	0.5	7	100.0	64/64	100.0	64	72.0	311/432	72.0	311

Table 41: Hybrid ELGAR experimental results for the MONK's Problem 2 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	5.368	30.0	375	95.4	116/122	95.9	117	96.5	417/432	97.2	420
22	0.923	36.6	382	96.1	117/122	96.7	118	94.6	409/432	97.2	420
23	5.367	26.5	344	95.4	116/122	95.9	117	96.7	418/432	97.2	420
24	5.368	30.0	375	95.4	116/122	95.9	117	96.5	417/432	97.2	420
25	5.367	26.2	344	95.4	116/122	95.9	117	96.7	418/432	97.2	420
26	5.270	6.2	88	94.6	115/122	95.9	117	97.8	422/432	100.0	432
27	5.211	5.8	82	94.1	115/122	95.1	116	97.7	422/432	99.5	430
28	5.501	1024.3	454	96.6	118/122	99.2	121	95.1	411/432	97.2	420
29	5.501	997.3	454	96.6	118/122	99.2	121	95.1	411/432	97.2	420
30	5.916	0.1	3	100.0	27/27	100.0	27	98.8	427/432	100.0	432

Table 42: Hybrid ELGAR experimental results for the MONK's Problem 3 Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	5.741	1299.7	324	99.8	4053/4062	99.8	4053	99.8	4052	99.8	4052
23	5.741	1558.0	430	99.8	4053/4062	99.8	4053	99.8	4052	99.8	4052
24	5.741	1201.4	325	99.8	4053/4062	99.8	4053	99.8	4052	99.8	4052
26	5.737	58.0	15	99.7	4052/4062	99.8	4053	99.7	4051	99.8	4052
27	5.567	101.6	30	98.3	3994/4062	99.8	4052	98.4	3998	99.7	4050
30	5.768	0.1	1	100.0	56/56	100.0	56	99.8	4052	99.8	4052

Table 43: Hybrid ELGAR experimental results for the Mushrooms Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	1.806	237.3	306	66.5	832/1252	66.5	832	64.2	804/1252	64.3	805
23	1.861	251.2	316	66.9	838/1252	68.8	862	64.4	806/1252	64.9	813
24	1.954	364.2	373	67.8	848/1252	71.7	898	64.4	806/1252	64.9	812
26	1.807	13.2	15	66.5	832/1252	66.7	835	64.1	803/1252	64.3	805
27	1.920	21.8	24	67.5	845/1252	68.9	863	64.6	809/1252	65.5	820
30	5.719	3.4	10	100.0	421/421	100.0	421	64.4	806/1252	64.4	806

Table 44: Hybrid ELGAR experimental results for the Ludwig Trial V 17-Attribute Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	1.843	336.9	323	66.8	836/1252	66.8	836	63.9	800/1252	64.3	805
23	1.842	365.2	342	66.8	836/1252	66.8	836	63.8	799/1252	64.3	805
24	1.834	269.4	318	66.7	835/1252	66.7	835	63.9	801/1252	64.3	805
26	1.825	14.4	14	66.6	834/1252	66.7	835	64.1	802/1252	64.3	805
27	1.826	27.6	28	66.6	834/1252	66.7	835	64.0	802/1252	64.3	805
30	5.719	5.5	13	100.0	421/421	100.0	421	64.1	802/1252	64.3	805

Table 45: Hybrid ELGAR experimental results for the Ludwig Trial V 31-Attribute Data Set

Exp.	Fit.	CPU	Gen.	Avg. Train		Max. Train		Avg. Recall		Max. Recall	
				%	#	%	#	%	#	%	#
21	4.328	81.8	374	98.5	345/350	98.6	345	95.1	332/349	96.0	335
22	0.969	95.6	427	98.5	345/350	98.6	345	94.7	331/349	96.0	335
23	4.376	79.5	355	98.9	346/350	100.0	350	93.4	326/349	95.1	332
24	4.328	81.9	374	98.5	345/350	98.6	345	95.1	332/349	96.0	335
25	4.376	78.6	355	98.9	346/350	100.0	350	93.4	326/349	95.1	332
26	4.212	5.3	21	97.5	341/350	98.0	343	94.2	329/349	94.8	331
27	4.164	4.0	16	97.1	340/350	97.4	341	91.8	320/349	93.4	326
28	4.503	220.1	36	100.0	350/350	100.0	350	92.8	324/349	93.7	327
29	4.410	91.6	18	99.2	347/350	100.0	350	94.3	329/349	95.4	333
30	4.513	0.6	24	100.0	24/24	100.0	24	92.1	322/349	92.8	324

Table 46: Hybrid ELGAR experimental results for the Wisconsin Breast Cancer Data Set

Table 47 lists the best result obtained for each data set using Hybrid ELGAR, along with results for the best one-rule found, and the best result obtained from ELGAR. For result from ELGAR and Hybrid ELGAR, the number of the experiment(s) which produced it is also shown. The 'Baseline' column indicates the baseline accuracy for the recall set in each case.

Data Set	Baseline	Results					
		Genetic ELGAR	Exp.	One-Rule	Hybrid ELGAR	Exp.	C4
Ljubljana	73.7%	77.9%	5	72.6%	82.1%	24 - 25	76.8%
MONK's 1	50.0%	100.0%	2 - 18	75.0%	100.0%	21 - 25	75.9%
MONK's 2	67.1%	84.5%	20	67.1%	79.4%	28, 29	65.1%
MONK's 3	52.8%	100.0%	17 - 20	80.6%	100.0%	26, 30	97.2%
Mushrooms	51.3%	99.8%	2	98.4%	99.8%	21 - 26, 30	100.0%
Trial V/17	53.6%	64.8%	2	64.3%	65.3%	27	65.0%
Trial V/31	53.6%	53.6%	1, 2	64.3%	64.3%	all	61.7%
Wisconsin	66.2%	96.8%	6 - 10	92.3%	96.0%	21, 22, 24	93.7%

Table 47: Hybrid ELGAR results compared with one-rule, ELGAR and C4

5.3 Conclusions from Hybrid ELGAR

Comparison of the results from Hybrid ELGAR with those of ELGAR suggests the following:

- In all but two data sets, Hybrid ELGAR performs as well as or better than ELGAR. The degree of improvement is greatest in the case of the 31-attribute version of the Ludwig Trial V data set, somewhat less for the Ljubljana breast cancer data set, and very small in the remaining cases.

- Surprisingly, Hybrid ELGAR performs worse than ELGAR in the MONK'S 2 and Wisconsin breast cancer data sets. Analysis of the detailed output from these data sets suggests different causes in each case.

  - The best possible one-rule for the MONK's 2 problem turns out to be the baseline rule for the data set, namely

    TRUE → not member

This rule necessarily contains zero bits corresponding to all possible values of one attribute (in this case, the first attribute, **a1**), and is sufficient to cause the search to find a local minimum.

- The Wisconsin breast cancer data set has the property that, for every attribute, there is a one-rule which has much greater accuracy than the baseline for the data set. The best one-rule overall, in fact, has an accuracy of 92.3% for the recall set, and 93.1% for the training set. This is enough to lead the hybrid to 97% training accuracy or better in every pass of every run. Unfortunately, in general this high training accuracy appears to be achieved at the expense of accuracy on the recall set, a symptom which is typical either of overfitting or of **oversearching** (defined in [Quinlan95] as "the discovery by extensive search of a theory that is not necessarily overcomplex but whose apparent accuracy is also misleading").

- As with ELGAR, no one experimental design is consistently superior for all data sets. However, the following results appear to hold:

  - Attribute filtering is useful. As evidence, experiment 24 performs as well as or better than experiment 21 in all data sets.

  - Attribute reordering appears to be useful, although not in all cases. In particular, experiment 23 performs as well as or better than experiment 21 in seven of eight data sets (the exception being the Wisconsin breast cancer data set); likewise, experiments 25 and 24 compare in a similar manner.

  - Like the proverbial tale of the young child learning to spell the word 'banana', the difficulty with genetic search, either alone or as part of the hybrid algorithm, is knowing when to stop. This is demonstrated by the comparison of experiments 24 and 26, which differ only in that experiment 26 tests for convergence as a termination criterion.

    In all but the MONK's 3 data set, experiment 24 required significantly more CPU time, but gave better performance. In the case of MONK's 3, experiment 26 terminated earlier than experiment 24 due to the convergence test, and as a result avoided the oversearching problem that caused

experiment 24 to perform better on the training set but worse on the recall set.

- In ELGAR, the original McCallum and Spackman version of overpopulation [McCallum90] appeared to be more useful than the modified version, particularly in combination with a richer set of operators. Hybrid ELGAR appears to reduce the magnitude of this effect, as shown by the fact that experiments 28 and 29 outperform the other experiments on only one data set (specifically, MONK's 2).

- The results of experiment 30 suggest that *meta-rule mode* searches are moderately successful, but not outstandingly so. This experiment is in the middle of the pack in terms of recall set accuracy on six of the eight data sets, while tied for the lead on MONK's 3 and dead last on the MONK's 2 and Wisconsin breast cancer data sets. Note that the apparently dramatic reduction in CPU time required by this experiment is misleading, since the average CPU time shown only considers the time spent learning those training instances which were not covered by the best one-rule found. The actual time required for this run was similar to that of the other experiments.

- Overall, Hybrid ELGAR is even more competitive with C4 than ELGAR, outperforming it in seven of eight data sets. Again, Hybrid ELGAR does particularly well compared with C4 on the MONK's problems, but this time its performance on both versions of the Ludwig Trial V data set exceeds that of C4, albeit by a thin margin.

- [Holte93, p. 67] makes this comment concerning the Ljubljana breast cancer data set (which Holte denotes as "BC"):

  > For example, on two datasets (BC, HE), few learning systems have succeeded in finding rules of any kind whose accuracy exceeds the baseline accuracy by more than 2 percentage points.

  Considering that Hybrid ELGAR achieves 82.1% accuracy compared to the 73.7% baseline accuracy for this data set, this comment, along with the statistics on which it was based, are particularly interesting.

- The data representation format remains an issue, but the results of the experiments involving attribute reordering suggest either that epistasis (*i.e.* the condition in which physically separate bits have mutually dependent optimal values; see Section 4.6) is simply is not a serious problem in the domains investigated here, or else that reordering attributes is not an effective technique to combat it.

Chapter 6

Conclusions and Future Directions

6.1 Concluding Remarks

The experimental results obtained with Hybrid ELGAR strongly suggest that the hybrid "simplicity first" genetic learning paradigm is a viable approach to concept learning, and that it performs better than ELGAR or Holte's 1R classifier [Holte93] alone. Room for improvement definitely exists, a topic which will be developed in Section 6.2; nevertheless the existing algorithm already exhibits performance which, for seven of eight data sets studied, equals or exceeds that of C4, itself widely held to be a benchmark for concept learning quality.

6.2 Suggestions for Further Research

Many promising directions exist for extending the work performed in this thesis. The following list includes some of the more interesting ones.

- A central issue in the design of any concept learner is the concept description language it will use. Hybrid ELGAR expresses concepts in the modified k-DNF language used by GABIL, a dialect which is well suited to conversion to bit string form for use with genetic search. However, there are concepts which are difficult to express this way; typically these involve situations where the value of one attribute depends on that of another.

For example, the MONK's Problems are deliberately designed to be difficult to represent in DNF, in which rules such as

$$(a1 = a2) \rightarrow \texttt{member}$$

(where **a1** and **a2** are both attribute names) cannot be expressed directly; the number of rules required to represent them causes an increase in the length of highly fit individuals, which makes them much more difficult to create, and easier to lose via recombination (a consequence of the Building Block Hypothesis). A more powerful concept description language would therefore seem likely to improve Hybrid ELGAR's performance, assuming it could be properly translated into a form suitable for genetic search.

- A key issue in any genetic learning system is that of data representation. The Pittsburgh-style representation format adopted in Hybrid ELGAR works reasonably well, but suffers from the drawback that it is impossible to predict *a priori* whether the order of attributes within each rule might lead to epistasis. Hybrid ELGAR's approach of reordering attributes according to their 1R ranking appears to give better performance in some data sets and worse performance in others, which suggests that attribute ordering is not a general solution, or else that 1R ranking order is not necessarily optimal. It is not at all obvious how to design a data representation in which attribute ordering would become irrelevant, but it seems likely that such a design would be highly desirable.

- Genetic search tends to be remarkably robust in the face of minor changes to the algorithm itself, which suggests that tuning the genetic search component in Hybrid ELGAR is unlikely to be helpful. However, there is certainly scope to develop better learning-related genetic operators. Janikow has successfully explored this direction [Janikow91], and more of his ideas might well be helpful for Hybrid ELGAR.

- Genetic search is particularly sensitive to the tradeoff between CPU time and solution quality. A better understanding of the nature of the CPU time vs. classification accuracy graph would give the user much greater control over Hybrid ELGAR's behaviour. While a theoretical model of this graph would likely be very difficult to create, an approximation might be possible in the form of a more accurate test for convergence. The test currently used in Hybrid

ELGAR, which compares the average fitness of the current generation to that of the most fit individual found to date, certainly has the property of limiting the CPU time spent in genetic search; however, in practice it occasionally causes the search to terminate too early. One possible test which might be more accurate, albeit more time-consuming itself, would be to compare the members of the current generation on a bit-by-bit basis, considering a given bit position to have converged if the same value is found in that position in more than some threshold percentage of the population; in this model, the population would be considered to have converged when a predetermined percentage of bit positions had converged.

- The most computationally expensive component of genetic search is the fitness function, which involves evaluating every candidate rule set against every training instance; this is proportional both to the population size and to the size of the training set. Any technique which could reduce the number of function evaluations without losing accuracy would therefore be helpful. This question requires a theoretical framework; practical techniques such as caching the results of previous evaluations are unhelpful due to the large number of them involved — this has been tried, and in one experiment involving the Ludwig Trial V 17-attribute subset, the amount of memory used by that variant of ELGAR exceeded the per-process virtual memory limit of 1024 megabytes on a DEC AlphaServer 2100 5/250.

- Genetic search generally tends to be good at finding the neighbourhood of the global optimum, but does not guarantee to find the optimum itself. Other hill-climbing methods, such as neural networks, generally exhibit the reverse behaviour, in that they can almost always find the global optimum from a position somewhere in its neighbourhood, but their locality of reference often makes it difficult for them to locate that neighbourhood in the first place. Given these complementary characteristics, it seems natural to perform a genetic search for a limited number of generations, and then use the best rule set found to design an appropriate neural network, in the manner of KBANN [Towell90, Towell93].

- Due to the influence of GABIL, Hybrid ELGAR is designed around the Pittsburgh approach to genetic learning. It would be interesting to learn how it would perform using a Michigan-style population along with McCallum and Spackman's "example sharing" technique [McCallum90].

- Finally, one strength of the genetic algorithm is that it does not require domain knowledge. However, if there happens to be domain knowledge available, it might be interesting to modify the fitness function to take it into account; for example, incentives might be given to rule sets which are consistent with an existing domain theory, and/or penalties applied to rule sets which are inconsistent with it.

References

[Bala94] Jerzy W. Bala, Kenneth A. De Jong, and Peter W. Pachowicz. Multistrategy Learning from Engineering Data by Integrating Inductive Generalization and Genetic Algorithms. In Ryszard S. Michalski and Gheorge Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume IV, chapter 18, pages 471–487. Morgan Kaufmann, San Francisco, California, 1994.

[Beasley93] David Beasley, David R. Bull, and Ralph R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[Bonelli90] Pierre Bonelli, Alexandre Parodi, Sandip Sen, and Stewart W. Wilson. NEWBOOLE: A Fast GBML System. In Bruce Porter and Raymond Joseph Mooney, editors, *Machine Learning: Proceedings of the Seventh International Conference (1990)*, pages 153–159, San Mateo, California, 1990. Morgan Kaufmann.

[Cantú-Paz97a] Erick Cantú-Paz. Designing Efficient Master-Slave Parallel Genetic Algorithms. Technical report, University of Illinois at Urbana-Champaign, 1997. IlliGAL Report No. 97004.

[Cantú-Paz97b] Erick Cantú-Paz. A Survey of Parallel Genetic Algorithms. Technical report, University of Illinois at Urbana-Champaign, 1997. IlliGAL Report No. 97003.

[Caudill92] Maureen Caudill and Charles Butler. *Understanding Neural Networks: Computer Explorations*, volume 1. The MIT Press, Cambridge, Massachusetts, 1992.

[Cavalli88] F. Cavalli, A. Goldhirsch, and R. Gelber. The Ludwig Breast Cancer Studies: Adjuvant Therapy for Early Breast Cancer. In V. Craig Jordan, editor, *Estrogen/antiestrogen Action and Breast Cancer Therapy*, pages 459–469. University of Wisconsin Press, Madison, Wisconsin, 1988.

[Clark90] P. Clark. Machine Learning: Techniques and Recent Developments. In A. P. Mirzai, editor, *Artificial Intelligence: Concepts and Applications in Engineering*, pages 65–93. Chapman & Hall, London, U.K, 1990.

[Cost93] Scott Cost and Steven Salzberg. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10(1):57–78, 1993.

[Davis89] L. Davis. Adapting Operator Probabilities in Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.

[De Jong75] Kenneth A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Ph.D. thesis, University of Michigan, Ann Arbor, 1975.

[De Jong90] Kenneth A. De Jong. Genetic Algorithm-Based Learning. In Yves Kodratoff and Ryszard S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume III, chapter 21, pages 611–638. Morgan Kaufmann, San Mateo, California, 1990.

[De Jong93] Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using Genetic Algorithms for Concept Learning. *Machine Learning*, 13(2/3):161–188, 1993.

[De Jong97] Kenneth A. De Jong, Mitchell A. Potter, and William M. Spears. Using Problem Generators to Explore the Effects of Epistasis. In *Proceedings of the Seventh International Conference On Genetic Algorithms*, pages 338–345. Morgan Kaufmann, 1997.

[Eshelman91] Larry J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 265–283. Morgan Kaufmann, 1991.

[Giarratano94] Joseph Giarratano and Gary Riley. *Expert Systems: Principles and Programming.* PWS Publishing Company, Boston, MA, 1994.

[Giguère98] Philippe Giguère and David Edward Goldberg. Population Sizing for Optimum Sampling with Genetic Algorithms: A Case Study of the Onemax Problem. In *Genetic Programming: Proceedings of the Third Annual Conference.* Morgan Kaufmann, 1998.

[Goldberg89] David Edward Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, MA, 1989.

[Goldberg91] David Edward Goldberg and Kalyanmoy Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 69–93. Morgan Kaufmann, 1991.

[Goldberg94] David Edward Goldberg. First Flights at Genetic-Algorithm Kitty Hawk. Technical report, University of Illinois at Urbana-Champaign, 1994. IlliGAL Report No. 94008.

[Goldberg98] David Edward Goldberg. The Design of Innovation: Lessons from Genetic Algorithms, Lessons for the Real World. Technical report, University of Illinois at Urbana-Champaign, 1998. IlliGAL Report No. 98004.

[Grefenstette86] John J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-16(1):122–128, 1986.

[Group89] The Ludwig Breast Cancer Study Group. Prolonged Disease-Free Survival After One Course of Perioperative Adjuvant Chemotherapy for Node-Negative Breast Cancer. *The New England Journal of Medicine*, 320(8):491–495, 1989.

[Holland75] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1975.

[Holmes95] Geoffrey Holmes and Craig G. Nevill-Manning. Feature Selection via the Discovery of Simple Classification Rules. In *Proceedings of the Symposium on Intelligent Data Analysis (IDA-95)*, pages 75–79, Baden-Baden, Germany, 1995.

[Holte93] Robert C. Holte. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11(1):63–90, 1993.

[Janikow91] Cezary Z. Janikow. *Inductive Learning of Decision Rules from Attribute-Based Examples: A Knowledge-Intensive Genetic Algorithm Approach.* Ph.D. thesis, University of North Carolina at Chapel Hill, 1991.

[Kamber91] Micheline Kamber. Automated Detection of Multiple Sclerosis Lesions in Magnetic Resonance Images of the Human Brain. M.Comp.Sci thesis, Concordia University, Montreal, Quebec, Canada, 1991.

[Kamber94] Micheline Kamber. Knowledge Base for the Ludwig Breast Cancer Database, Trial V. Based in part on notes from Karen Price, 1994.

[Kazadi98] Sanza T. Kazadi. Conjugate Schema and the Basis Representation of Crossover and Mutation Operators. *Evolutionary Computation*, 6(2):129–160, 1998.

[Kullback59] S. Kullback. *Information Theory and Statistics.* John Wiley and Sons, Inc., 1959.

[Louis93] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design.* Ph.D. thesis, Indiana University, 1993.

[Marill63] Marill and Green. On the Effectiveness of Receptors in Recognition Systems. *IEEE Transactions on Information Theory*, 9:11–17, January 1963.

[McCallum90] R. Andrew McCallum and Kent A. Spackman. Using Genetic Algorithms to Learn Disjunctive Rules from Examples. In Bruce Porter and Raymond Joseph Mooney, editors, *Machine Learning: Proceedings of the Seventh International Conference (1990)*, pages 149–152, San Mateo, California, 1990. Morgan Kaufmann.

[Merz92] C. J. Merz and P. M. Murphy, Documentation for the Wisconsin Breast Cancer Data Set from the UCI Repository, 1992, University of California at Irvine, Department of Information and Computer Sciences,

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names.

[Merz93] C. J. Merz and P. M. Murphy, Documentation for the MONK's Problems Data Set from the UCI Repository, 1993, University of California at Irvine, Department of Information and Computer Sciences,

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/monks-problems/monks.names.

[Merz95] C. J. Merz and P. M. Murphy, Documentation for the Ljubljana Breast Cancer Data Set from the UCI Repository, 1995, University of California at Irvine, Department of Information and Computer Sciences,

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer/breast-cancer.names.

[Merz97] C. J. Merz and P. M. Murphy, Documentation for the Mushrooms Data Set from the UCI Repository, 1997, University of California at Irvine, Department of Information and Computer Sciences,

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom/agaricus-lepiota.names.

[Merz98] C. J. Merz and P. M. Murphy, UCI Repository of Machine Learning Databases, 1998, University of California at Irvine, Department of Information and Computer Sciences,

http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Michalewicz92] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1992.

[Michalski83] Ryszard S. Michalski. A Theory and Methodology of Inductive Learning. In Ryszard S. Michalski, J. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Tioga Publishing, Palo Alto, California, 1983.

[Michalski86] Ryszard S. Michalski. The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045. Morgan Kaufmann, 1986.

[Michalski90] Ryszard S. Michalski. Learning Flexible Concepts: Fundamental Ideas and a method based on Two-Tiered Representation. In Yves Kodratoff and Ryszard S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume III, chapter 3, pages 63–102. Morgan Kaufmann, San Mateo, California, 1990.

[Miller95] Brad L. Miller and David Edward Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. Technical report, University of Illinois at Urbana-Champaign, 1995. IlliGAL Report No. 95006.

[Mitchell80] Tom M. Mitchell. The Need for Biases in Learning Generalizations. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*, pages 185–191. Morgan Kaufmann, San Mateo, California, 1980.

[Nevill-Manning95] Craig G. Nevill-Manning, Geoffrey Holmes, and Ian H. Witten. The Development of Holte's 1R Classifier. In Nikola K. Kasabov and George Coghill, editors, *ANNES'95 The Second New Zealand International Two-Stream Conference on Artificial Neural Networks & Expert Systems*, pages 239–242, Dunedin, NZ, 1995. University of Otago.

[Opitz97] David W. Opitz and Jude W. Shavlik. Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies. *Journal of Artificial Intelligence Research*, 6:177–209, 1997.

[Patton98] Arnold L. Patton, Terrence Dexter, Erik D. Goodman, and William F. Punch, III. On the Application of Cohort-Driver Operators to Continuous Optimization Problems using Evolutionary Computation. In *Proceedings of the Seventh Conference on Evolutionary Computation*. Springer Verlag, 1998.

[Pawlowsky92] Marc Andrew Pawlowsky. Modified Uniform Crossover and Desegregation in Genetic Algorithms. M.Comp.Sci thesis, Concordia University, Montreal, Quebec, Canada, 1992.

[Pawlowsky95] Marc Andrew Pawlowsky. Crossover Operators. In Lance Chambers, editor, *Practical Handbook of Genetic Algorithms*, volume 1, pages 101–114. CRC Press, 1995.

[Quinlan79] J. Ross Quinlan. Discovering Rules by Induction from Large Collections of Examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*, pages 168–201. Edinburgh University Press, UK, 1979.

[Quinlan86] J. Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[Quinlan87] J. Ross Quinlan. Simplifying Decision Trees. *International Journal of Man Machine Studies*, 27:221–234, 1987.

[Quinlan90] J. Ross Quinlan. Probabilistic Decision Trees. In Yves Kodratoff and Ryszard S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume III, chapter 5, pages 140–152. Morgan Kaufmann, San Mateo, California, 1990.

[Quinlan93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

[Quinlan95] J. Ross Quinlan and R. M. Cameron-Jones. Oversearching and Layered Search in Empirical Learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. Morgan Kaufmann, 1995.

[Quinlan96] J. Ross Quinlan. Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Schaffer87] J. David Schaffer and A. Morishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 36–40. Lawrence Erlbaum, 1987.

[Schlimmer87] Jeffrey S. Schlimmer. *Concept Acquisition Through Representational Adjustment.* Ph.D. thesis, University of California at Irvine, 1987.

[Shavlik86] Jude W. Shavlik. Learning Classical Physics. In Tom M. Mitchell, J. G. Carbonell, and Ryszard S. Michalski, editors, *Machine Learning: A Guide to Current Research*, pages 307–310. Kluwer Academic, 1986.

[Shavlik90] Jude W. Shavlik and Thomas G. Dietterich, editors. *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, California, 1990.

[Spears98] William M. Spears. *The Role of Mutation and Recombination in Evolutionary Algorithms.* Ph.D. thesis, George Mason University, 1998.

[Stanfill86] C. Stanfill and D. Waltz. Toward Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[Thornton92] C. J. Thornton. *Techniques in Computational Learning.* Chapman & Hall, London, 1992.

[Thrun91] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's Problems – A Performance Comparison of Different Learning Algorithms. Technical Report CS-CMU-91-197, Carnegie Mellon University, 1991.

[Towell90] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866. AAAI Press, 1990.

[Towell93] Geoffrey G. Towell and Jude W. Shavlik. Extracting Refined Rules from Knowledge-Based Neural Networks. *Machine Learning*, 13(1):71–101, 1993.

[Tu92] Pei-Lei Tu and Jen-Yao Chung. A New Decision-Tree Classification Algorithm for Machine Learning. In *Proceedings of the 4th International Conference On Tools with Artificial Intelligence*, pages 370–377. IEEE Computer Society Press, 1992.

[Tuson98] Andrew Tuson and Peter Ross. Adapting Operator Settings in Genetic Algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.

[Vafaie92] Haleh Vafaie and Kenneth A. De Jong. Genetic Algorithms as a Tool for Feature Selection in Machine Learning. In *Proceedings of the 4th International Conference On Tools with Artificial Intelligence*, pages 200–203. IEEE Computer Society Press, 1992.

[Vafaie94] Haleh Vafaie and Kenneth A. De Jong. Improving a Rule Induction System Using Genetic Algorithms. In Ryszard S. Michalski and Gheorge Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume IV, chapter 17, pages 453–469. Morgan Kaufmann, San Francisco, California, 1994.

[Vose98a] Michael D. Vose and Alden H. Wright. The Simple Genetic Algorithm and the Walsh Transform: Part I, Theory. *Evolutionary Computation*, 6(3):253–273, 1998.

[Vose98b] Michael D. Vose and Alden H. Wright. The Simple Genetic Algorithm and the Walsh Transform: Part II, The Inverse. *Evolutionary Computation*, 6(3):275–289, 1998.

[Whitley93] Darrell Whitley. A Genetic Algorithm Tutorial. Technical Report CS-93-103, Colorado State University, 1993.

[Wilson87] Stewart W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning*, 2:199–228, 1987.

[Wolberg90] William H. Wolberg and Olvi L. Mangasarian. Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology. *Proceedings of the National Academy of Sciences*, 87:9193–9196, 1990.

[Zhang94] Jianing Zhang. Learning Graded Concept Descriptions by Integrating Symbolic and Subsymbolic Approaches. In Ryszard S. Michalski and Gheorge Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume IV, chapter 16, pages 431–452. Morgan Kaufmann, San Francisco, California, 1994.

Appendix A

Glossary

Note that terms in SMALL CAPS are cross-references to other entries within this glossary.

1-rule see ONE-RULE

antecedent the left-hand side of a rule; *e.g.* in '$a \wedge b \rightarrow c$', the antecedent is '$a \wedge b$'

attribute an observable characteristic of an INSTANCE which is used in classification; *e.g.* if the CONCEPT to be learned is LION, then some possible attributes are **age**, **size**, **skin type**, and **colour**; an attribute has two or more potential VALUES; attributes are also referred to as *features*

baseline accuracy the CLASSIFICATION ACCURACY of the default rule in a TRAINING SET or RECALL SET, *i.e.* the classification accuracy obtained by guessing that all INSTANCES in the set are members of the set's most common class

bias the tendency of a learning program to prefer certain potential CONCEPT descriptions over others; this will be influenced by the language in which concept descriptions are expressed, as well as the specific method by which the program attempts to learn

binary tournament selection a method of SELECTION used with the GENETIC ALGORITHM, in which two INDIVIDUALS are chosen at random from the POPULATION, and the one with the greater FITNESS is added to the MATING POOL

building blocks SCHEMATA which have a short DEFINING LENGTH, low ORDER and above-average FITNESS

children see OFFSPRING

classification accuracy the percentage of RECALL INSTANCES which are correctly classified by a candidate CONCEPT description

concept (i) the goal of the learning process, *i.e.* a specific idea to be learned
(ii) a specific rule set, decision tree or other formulation created by a learning program as a candidate to represent the actual concept to be learned

concept learning the process of creating the best possible representation of a CONCEPT based on a set of TRAINING INSTANCES

conjunct one of the terms in a conjunction; *e.g.* the ANTECEDENT of the rule '(a = x) ∧ (b = y) → c' contains two conjuncts, namely '(a = x)' and '(b = y)'

consequent the right-hand side of a rule; *e.g.* in 'a ∧ b → c', the consequent is 'c'

continuous adjective which describes an ATTRIBUTE that may assume any numerical value in a specific, possibly unbounded range, as opposed to being restricted to one of a finite set of predetermined names or numbers (*e.g.* an attribute **temperature**, the value of which can be any real number greater than or equal to -273); an attribute which is not continuous is said to be DISCRETE or NOMINAL

convergence a GENETIC SEARCH is said to have converged when a majority of the POPULATION are copies or near copies of the most highly fit INDIVIDUAL in the population; at this point further improvement becomes unlikely, because most CROSSOVER operations will produce OFFSPRING identical to their PARENTS

convergence, premature see PREMATURE CONVERGENCE

cover a rule is said to cover an INSTANCE if the rule's ANTECEDENT becomes a true statement when the ATTRIBUTE values from the instance are substituted into it; *e.g.* the rule '**(size = large) ∧ (colour = red ∨ blue) → widget**' would cover an instance which is **large** and **red**, but not one which is **small** and **red** or one which is **large** and **green**

crossover a GENETIC OPERATOR which creates two OFFSPRING by copying complementary portions of the bit strings from each of two PARENTS

defining length (of a schema) the distance between the first and last fixed bit positions in a SCHEMA

discrete adjective which describes an ATTRIBUTE that has a restricted, enumerated set of possible VALUES, as opposed to being free to assume any numerical value in a specific range (*e.g.* an attribute **colour**, the value of which must be `red`, `green`, `yellow` or `blue`); the term NOMINAL is occasionally used as a synonym; an attribute which is not discrete is said to be CONTINUOUS

disjunctive normal form a format in which a logical formula is written as a disjunction of conjunctions, *e.g.*

$$(A = t \wedge B = u) \vee (C = v \wedge D = w \wedge E = x) \vee (F = y \wedge G = z)$$

when the number of CONJUNCTS per conjunction must not exceed an arbitrary constant, the resulting form is called *k-DNF*

DNF see DISJUNCTIVE NORMAL FORM

epistasis a condition that can occur in GENETIC SEARCH, in which bits that are closely related (in the sense that they are part of the same BUILDING BLOCK) are widely separated physically within a bit string; this condition causes building blocks to have long DEFINING LENGTHS, making them vulnerable to disruption

error rate the percentage of RECALL INSTANCES classified incorrectly by a candidate CONCEPT (*e.g.* a rule set or decision tree)

example see INSTANCE

feature see ATTRIBUTE

fitness an arbitrary measurement of the worth of a given INDIVIDUAL as a candidate solution in GENETIC SEARCH

fitness function a function which computes the FITNESS of an INDIVIDUAL during GENETIC SEARCH; the quality of the solution found by the search will depend on how accurately the fitness function represents the actual worth of an arbitrary individual

generalization the process of changing a rule to make it more general, *i.e.* to allow it to COVER a greater number of INSTANCES

generation a specific POPULATION at a particular moment in time during a GENETIC SEARCH; each generation is succeeded by a new one at each iteration of the main loop of the search process

genetic algorithm an algorithm designed by John H. Holland [Holland75] to solve search and optimization problems by means of the simultaneous application of RECOMBINATION techniques and selection pressure

genetic operators operations such as CROSSOVER or MUTATION, which map one or more INDIVIDUALS onto one or more new individuals, and which are used during the RECOMBINATION step of the GENETIC ALGORITHM

genetic search any search process based on the GENETIC ALGORITHM

individual a single candidate solution in GENETIC SEARCH, typically one of many in the POPULATION

instance an object which may or may not belong to the CONCEPT being learned; typically a learning algorithm is presented with a TRAINING SET containing instances which are correctly classified, from which it will attempt to learn the concept; instances are also referred to as *examples*

internal disjunction a disjunction of VALUES within a CONJUNCT; *e.g.* in the rule '(a = x) ∧ (b = y ∨ z) → c', the term '(a = x)' is a normal conjunct, but '(b = y ∨ z)' contains the internal disjunction 'y ∨ z'

k-DNF see DISJUNCTIVE NORMAL FORM

mating pool the collection of INDIVIDUALS chosen by the SELECTION step of a GENETIC ALGORITHM, from which PARENTS are randomly drawn to be used as input to the CROSSOVER and MUTATION operators

mutation a GENETIC OPERATOR which creates one OFFSPRING by randomly inverting a bit from a PARENT individual

negative instance an INSTANCE which is not a member of the CONCEPT being learned

noise the presence of incorrectly classified INSTANCES in a TRAINING SET

nominal see DISCRETE

offspring one or more INDIVIDUALS created by applying a GENETIC OPERATOR (*e.g.* CROSSOVER or MUTATION) to one or more existing PARENT individuals

one-rule a rule which is limited to tests involving only one ATTRIBUTE; a one-rule corresponds to a decision tree containing only one non-leaf node

operators see GENETIC OPERATORS

order (of a schema) the number of fixed bit positions in a SCHEMA, *i.e.* the number of positions which do not contain the * symbol

overfitting the creation of a CONCEPT description which relies on the details of specific TRAINING INSTANCES to such an extent that it exhibits poor CLASSIFICATION ACCURACY on RECALL INSTANCES

oversearching a term coined by J. Ross Quinlan [Quinlan95] to describe "the discovery by extensive search of a theory that is not necessarily overcomplex but whose apparent accuracy is also misleading"

parents INDIVIDUALS supplied as input to a GENETIC OPERATOR (*e.g.* CROSSOVER or MUTATION) in order to create one or more new individuals, which are termed OFFSPRING

population the collection of candidate solutions maintained by a GENETIC ALGORITHM; the members of a population are called INDIVIDUALS

positive instance an INSTANCE which is a member of the CONCEPT being learned

prediction accuracy see CLASSIFICATION ACCURACY

premature convergence the state when GENETIC SEARCH has CONVERGED too quickly; this is usually caused by the presence of an INDIVIDUAL which is not in the neighbourhood of the global optimum, but which nevertheless has sufficiently high FITNESS to dominate the POPULATION; when this occurs, the result of the search will usually be a local optimum rather than the global optimum

recall instance an INSTANCE which is to be classified after the learning process has terminated; if the correct classification for a recall instance is known, that instance can be used to evaluate the quality of the learned CONCEPT description

recall set a set of correctly classified RECALL INSTANCES used for the purpose of evaluating the quality of a learned CONCEPT description

recombination the process of applying any genetic OPERATOR which creates OFFSPRING by combining characteristics from two or more PARENTS; CROSSOVER is a typical operator of this kind; this is in contrast to genetic operators which create a single CHILD from only one parent, of which MUTATION is a typical example

reproduction the step in the GENETIC ALGORITHM in which INDIVIDUALS in the MATING POOL are combined using GENETIC OPERATORS in order to create the OFFSPRING which will comprise the next GENERATION

schema (plural SCHEMATA) a template which describes potential values for an INDIVIDUAL, by specifying the permissible value or values for every position in the individual's bit string

search space the set of all possible candidates which may be examined in a search; for CONCEPT LEARNING, the search space includes all possible CONCEPT descriptions

selection the step in the GENETIC ALGORITHM in which INDIVIDUALS are chosen, based on their FITNESS, for inclusion in the MATING POOL

self-adaptation a process by which a machine learning system can learn to adjust its own behaviour at the same time that it attempts to learn a CONCEPT

specialization the process of changing a rule to make it less general, *i.e.* to allow it to COVER a smaller number of INSTANCES

training instance an INSTANCE which has a known classification, provided as input to a CONCEPT LEARNING system

training set a set of TRAINING INSTANCES

values observations recorded for a given ATTRIBUTE in a given INSTANCE; *e.g.* for an attribute 'size', potential values might be 'small', 'medium' and 'large'; note that the specific values available for any given attribute depend on the design of the data set, which is usually decided at the time that data collection occurs

Appendix B

Hybrid ELGAR Output

As an illustration, the following is the complete output (verbatim except for line breaks to allow it to fit on the page) from experiment number 27 with Hybrid ELGAR, using the 17-attribute version of the Ludwig Trial V breast cancer data set.

```
================================================
ELGAR:  Evolutionary Learning by Genetic AlgoRithm
================================================

Operating Parameters
====================

population size      =   100
memory usage         =     1 byte per bit
generation limit     =   600
stagnation limit     =   300
number of attributes =    17
bits per rule        =    63
baseline accuracy    =    52.79553%  (training instances)
                     =    53.59425%  (recall   instances)
baseline rule        =   if TRUE then negative
```

- penalties WILL be imposed based on the number of rules per individual
- rewards WILL be given to individuals which exceed baseline accuracy
- "real" overpopulation will NOT be used
- instances with unknown values will ALWAYS be accepted
- pure genetic mode will NOT be used
- convergence tests WILL be performed
- attributes worse than baseline accuracy WILL be filtered out
- attributes WILL be sorted according to 1-rule 'usefulness'
- meta-rule mode will NOT be used

Instances
=========

- 2504 instances found (1252 train, 1252 recall)

Operator List
=============

name	applicable?
cross1	NO
cross2	YES
crossU	NO
crossMU	NO
crossR	NO
mutation	YES
addalt	NO
dropcond	NO
genspec	NO

Start Time
==========

Start of run = Sun Dec 6 14:31:11 1998

```
Exhaustive Search for 1-Rules
=============================

Sun Dec  6 14:31:12 1998:  examining attribute Treatment (4 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  57.90735% (725/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Nodes (4 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  66.37380% (831/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Menopause (3 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  52.79553% (661/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Age (7 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  55.43131% (694/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute ER Value (3 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  52.79553% (661/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute PR Value (3 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  52.79553% (661/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Tumour Size (2 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  56.70927% (710/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute MxPeri (5 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  49.44090% (619/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Leucovorin (2 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  52.79553% (661/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Path Grade (4 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  55.83067% (699/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Vessel Invasion (4 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  59.82428% (749/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Multicentric (2 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  55.83067% (699/1252 correct)
```

```
Sun Dec  6 14:31:12 1998:  examining attribute Histology (8 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  55.03195% (689/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Medullary tumour (2 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  53.99361% (676/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute CerbB2 Status (3 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  53.11502% (665/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Amenorrhea cbt (3 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  52.79553% (661/1252 correct)

Sun Dec  6 14:31:12 1998:  examining attribute Amenorrhea rec (4 bits)
Sun Dec  6 14:31:12 1998:  best accuracy found =  53.35463% (668/1252 correct)


Best 1-rule found
=================

 - attribute =    Nodes

 - CPU time  =     0.18000 seconds

 - fitness   =     1.79838

 - accuracy  =    66.37380% (831/1252 training instances correct)
             =    64.29713% (805/1252   recall instances correct)


 - unpruned rules
   --------------
   1111 0011 111 1111111 111 111 11 11111 11 1111 1111 11 11111111 11 111 111 1111

 - pruned rules
   ------------
   1111 0011 111 1111111 111 111 11 11111 11 1111 1111 11 11111111 11 111 111 1111
   ==>  (Nodes=3-4 OR Nodes=>4)
```

- confusion matrix:

		classified	
		positive	negative
actual:	positive	233	348
	negative	99	572

- instances which were classified incorrectly:

1254, 1256, 1266, 1269, 1271, 1272, 1273, 1274, 1276,
1278, 1279, 1281, 1282, 1284, 1287, 1288, 1290, 1291,
1292, 1293, 1295, 1296, 1297, 1299, 1301, 1310, 1313,
1315, 1319, 1321, 1324, 1327, 1331, 1334, 1336, 1337,
1340, 1342, 1343, 1344, 1348, 1354, 1372, 1375, 1382,
1385, 1386, 1387, 1389, 1393, 1395, 1398, 1399, 1400,
1403, 1411, 1415, 1416, 1420, 1421, 1423, 1424, 1425,
1428, 1429, 1434, 1439, 1440, 1445, 1448, 1453, 1454,
1465, 1467, 1471, 1475, 1478, 1485, 1497, 1503, 1506,
1507, 1512, 1513, 1514, 1518, 1520, 1521, 1525, 1527,
1532, 1538, 1551, 1554, 1555, 1556, 1557, 1558, 1559,
1560, 1561, 1566, 1567, 1568, 1584, 1586, 1588, 1590,
1591, 1592, 1596, 1597, 1598, 1599, 1600, 1601, 1602,
1604, 1606, 1610, 1615, 1621, 1622, 1624, 1626, 1636,
1637, 1641, 1642, 1645, 1652, 1653, 1658, 1659, 1660,
1667, 1671, 1673, 1676, 1684, 1685, 1686, 1690, 1694,
1699, 1704, 1708, 1709, 1710, 1712, 1714, 1715, 1716,
1720, 1721, 1722, 1724, 1731, 1737, 1738, 1740, 1744,
1746, 1748, 1752, 1753, 1754, 1757, 1759, 1761, 1763,
1765, 1768, 1770, 1772, 1773, 1775, 1778, 1782, 1786,
1795, 1796, 1802, 1804, 1805, 1810, 1811, 1813, 1814,
1815, 1818, 1819, 1827, 1830, 1831, 1833, 1834, 1839,
1844, 1845, 1846, 1848, 1851, 1852, 1853, 1857, 1858,
1859, 1867, 1870, 1871, 1872, 1876, 1880, 1882, 1886,
1892, 1893, 1894, 1898, 1899, 1900, 1909, 1912, 1914,
1915, 1918, 1919, 1920, 1921, 1924, 1926, 1927, 1930,
1933, 1934, 1936, 1937, 1939, 1941, 1942, 1945, 1946,

1949, 1951, 1953, 1955, 1961, 1963, 1968, 1972, 1973,
1979, 1982, 1983, 1999, 2000, 2002, 2006, 2015, 2016,
2017, 2018, 2020, 2021, 2023, 2024, 2025, 2026, 2027,
2028, 2030, 2031, 2034, 2038, 2039, 2041, 2042, 2043,
2045, 2047, 2052, 2053, 2054, 2057, 2058, 2061, 2066,
2067, 2068, 2069, 2071, 2079, 2081, 2083, 2088, 2089,
2090, 2091, 2092, 2093, 2098, 2102, 2104, 2106, 2108,
2117, 2118, 2119, 2123, 2127, 2130, 2132, 2138, 2143,
2146, 2149, 2151, 2157, 2163, 2166, 2167, 2168, 2171,
2174, 2175, 2176, 2177, 2179, 2181, 2182, 2184, 2187,
2189, 2190, 2191, 2192, 2194, 2196, 2207, 2208, 2213,
2215, 2216, 2220, 2222, 2223, 2224, 2230, 2238, 2241,
2242, 2244, 2249, 2250, 2251, 2254, 2257, 2261, 2268,
2269, 2273, 2274, 2282, 2283, 2290, 2296, 2297, 2299,
2300, 2302, 2304, 2305, 2307, 2313, 2314, 2318, 2321,
2322, 2327, 2334, 2335, 2339, 2342, 2346, 2350, 2351,
2352, 2353, 2354, 2355, 2361, 2362, 2365, 2374, 2378,
2379, 2380, 2381, 2382, 2383, 2390, 2394, 2395, 2399,
2400, 2402, 2407, 2408, 2413, 2414, 2416, 2417, 2418,
2421, 2423, 2426, 2427, 2428, 2434, 2435, 2437, 2440,
2441, 2442, 2444, 2445, 2446, 2449, 2450, 2452, 2453,
2454, 2457, 2460, 2465, 2467, 2468, 2470, 2478, 2480,
2481, 2483, 2487, 2491, 2499, 2501

Removing attributes which perform less well than baseline

==========

- removing attribute Amenorrhea cbt (3 bits)
- removing attribute Leucovorin (2 bits)
- removing attribute MxPeri (5 bits)
- removing attribute PR Value (3 bits)
- removing attribute ER Value (3 bits)
- removing attribute Menopause (3 bits)
- rebuilding 1-rules for 44-bit rule format

```
Sorting attributes in reverse order by performance
==================================================

 - new attribute order:

      0)  Nodes
      1)  Vessel Invasion
      2)  Treatment
      3)  Tumour Size
      4)  Path Grade
      5)  Multicentric
      6)  Age
      7)  Histology
      8)  Medullary tumour
      9)  Amenorrhea rec
     10)  CerbB2 Status


--------------------------------------------------------------------------------
START OF PASS NUMBER 1 OF 5 at Sun Dec  6 14:31:12 1998


Genetic Search Results
======================

*****  REACHED CONVERGENCE LIMIT:

   - generation number 14 at Sun Dec  6 14:31:50 1998
        - best fitness to date =     1.805
        - accuracy             =    66.454% (832/1252 correct)
        - correct              =   264 positive,   568 negative
        - incorrect            =   327 positive,    93 negative
        - average fitness      =     1.771
        - convergence factor   =     0.981
        - last improvement     = generation #14


--------------------------------------------------------------------------------
END OF PASS 1 at Sun Dec  6 14:31:50 1998 after 14 total generations
```

```
CPU usage for this pass =       12.730 seconds


Best individual found (at generation   14)
==========================================

 - fitness  =     1.80542

 - accuracy =    66.45367% (832/1252 training instances correct)
            =    64.05751% (802/1252 recall   instances correct)

 - length   =    88 bits =   2 rules at 44 bits/rule


 - unpruned rules
   --------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   0111 1101 0110 11 1111 01 0011111 01001101 11 1111 111

 - pruned rules
   ------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   ==>  (Nodes=3-4 OR Nodes=>4)
   0111 1101 0110 11 1111 01 0011111 01001101 11 1111 111
   ==>  Nodes<>0 AND Vessel Invasion<>2 AND
        (Treatment=NO RX OR Treatment=PECT+CONCT) AND Multicentric=1 AND
        (Age=41-45 OR Age=46-50 OR Age=51-55 OR Age=56-60 OR Age=>60) AND
        (Histology=1 OR Histology=4 OR Histology=5 OR Histology=7)


 - confusion matrix:
                                  classified
                                  ----------

                              positive   negative

                  positive      233        348
      actual:
                  negative      102        569
```

- instances which were classified incorrectly:

1254, 1256, 1266, 1269, 1271, 1272, 1273, 1274, 1276,
1278, 1279, 1281, 1282, 1284, 1287, 1288, 1290, 1291,
1292, 1293, 1295, 1296, 1297, 1299, 1301, 1310, 1313,
1315, 1319, 1321, 1324, 1327, 1331, 1334, 1336, 1337,
1340, 1342, 1343, 1344, 1348, 1354, 1372, 1375, 1382,
1385, 1386, 1387, 1389, 1393, 1395, 1398, 1399, 1400,
1403, 1411, 1415, 1416, 1420, 1421, 1423, 1424, 1425,
1428, 1429, 1434, 1439, 1440, 1445, 1448, 1453, 1454,
1465, 1467, 1471, 1475, 1478, 1485, 1497, 1503, 1506,
1507, 1512, 1513, 1514, 1518, 1520, 1521, 1525, 1527,
1532, 1538, 1551, 1554, 1555, 1556, 1557, 1558, 1559,
1560, 1561, 1566, 1567, 1568, 1584, 1586, 1588, 1590,
1591, 1592, 1596, 1597, 1598, 1599, 1600, 1601, 1602,
1604, 1606, 1610, 1615, 1621, 1622, 1624, 1626, 1636,
1637, 1641, 1642, 1645, 1652, 1653, 1658, 1659, 1660,
1667, 1671, 1673, 1676, 1684, 1685, 1686, 1690, 1694,
1699, 1704, 1708, 1709, 1710, 1712, 1714, 1715, 1716,
1720, 1721, 1722, 1724, 1731, 1734, 1737, 1738, 1740,
1744, 1746, 1748, 1752, 1753, 1754, 1757, 1759, 1761,
1763, 1765, 1768, 1770, 1772, 1773, 1775, 1778, 1782,
1786, 1795, 1796, 1802, 1804, 1805, 1810, 1811, 1813,
1814, 1815, 1818, 1819, 1827, 1830, 1831, 1833, 1834,
1839, 1844, 1845, 1846, 1848, 1851, 1852, 1853, 1857,
1858, 1859, 1867, 1870, 1871, 1872, 1876, 1880, 1882,
1886, 1892, 1893, 1894, 1898, 1899, 1900, 1909, 1912,
1914, 1915, 1918, 1919, 1920, 1921, 1924, 1926, 1927,
1930, 1933, 1934, 1936, 1937, 1939, 1941, 1942, 1945,
1946, 1949, 1951, 1953, 1955, 1961, 1963, 1968, 1972,
1973, 1979, 1982, 1983, 1999, 2000, 2002, 2006, 2015,
2016, 2017, 2018, 2020, 2021, 2023, 2024, 2025, 2026,
2027, 2028, 2030, 2031, 2034, 2038, 2039, 2041, 2042,
2043, 2045, 2047, 2052, 2053, 2054, 2057, 2058, 2061,
2066, 2067, 2068, 2069, 2071, 2079, 2081, 2083, 2088,
2089, 2090, 2091, 2092, 2093, 2098, 2102, 2104, 2106,
2108, 2117, 2118, 2119, 2123, 2127, 2130, 2132, 2138,
2143, 2146, 2149, 2151, 2157, 2163, 2165, 2166, 2167,
2168, 2171, 2174, 2175, 2176, 2177, 2179, 2181, 2182,

```
2184,  2187,  2189,  2190,  2191,  2192,  2194,  2196,  2207,
2208,  2213,  2215,  2216,  2220,  2222,  2223,  2224,  2230,
2238,  2241,  2242,  2244,  2249,  2250,  2251,  2254,  2257,
2261,  2268,  2269,  2273,  2274,  2282,  2283,  2290,  2296,
2297,  2299,  2300,  2302,  2304,  2305,  2307,  2313,  2314,
2316,  2318,  2321,  2322,  2327,  2334,  2335,  2339,  2342,
2346,  2350,  2351,  2352,  2353,  2354,  2355,  2361,  2362,
2365,  2374,  2378,  2379,  2380,  2381,  2382,  2383,  2390,
2394,  2395,  2399,  2400,  2402,  2407,  2408,  2413,  2414,
2416,  2417,  2418,  2421,  2423,  2426,  2427,  2428,  2434,
2435,  2437,  2440,  2441,  2442,  2444,  2445,  2446,  2449,
2450,  2452,  2453,  2454,  2457,  2460,  2465,  2467,  2468,
2470,  2478,  2480,  2481,  2483,  2487,  2491,  2499,  2501

--------------------------------------------------------------------------------
START OF PASS NUMBER 2 OF 5 at Sun Dec  6 14:31:50 1998


Genetic Search Results
======================

*****  REACHED CONVERGENCE LIMIT:

   - generation number 28 at Sun Dec  6 14:33:11 1998
       - best fitness to date =     2.004
       - accuracy             =    68.211% (854/1252 correct)
       - correct              =   305 positive,   549 negative
       - incorrect            =   286 positive,   112 negative
       - average fitness      =     1.976
       - convergence factor   =     0.986
       - last improvement     = generation #23


--------------------------------------------------------------------------------
END OF PASS 2 at Sun Dec  6 14:33:11 1998 after 28 total generations

CPU usage for this pass =       26.600 seconds
```

```
Best individual found (at generation   23)
==========================================

 - fitness  =     2.00381

 - accuracy =    68.21086% (854/1252 training instances correct)
            =    65.17572% (816/1252 recall   instances correct)

 - length   =   132 bits =   3 rules at 44 bits/rule


 - unpruned rules
   --------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   1001 1110 0001 10 0000 00 1000010 00011111 11 0110 001
   0100 1111 1100 11 1111 11 1101101 11111101 11 1111 111

 - pruned rules
   ------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   ==>  (Nodes=3-4 OR Nodes=>4)
   0100 1111 1100 11 1111 11 1101101 11111101 11 1111 111
   ==>  Nodes=1-2 AND (Treatment=PECT OR Treatment=NO RX) AND
        (Age=<31 OR Age=31-40 OR Age=46-50 OR Age=51-55 OR Age=>60) AND
        Histology<>6


 - confusion matrix:
                                    classified
                                    ----------

                               positive   negative

                   positive      270        311
      actual:
                   negative      125        546
```

- instances which were classified incorrectly:

1254, 1256, 1263, 1266, 1269, 1271, 1272, 1273, 1274,
1276, 1278, 1281, 1282, 1284, 1287, 1288, 1290, 1291,
1292, 1293, 1295, 1296, 1297, 1298, 1299, 1301, 1303,
1310, 1312, 1313, 1315, 1319, 1321, 1324, 1327, 1331,
1334, 1336, 1337, 1340, 1342, 1343, 1344, 1348, 1354,
1355, 1372, 1375, 1382, 1385, 1386, 1389, 1390, 1393,
1395, 1398, 1399, 1400, 1403, 1411, 1415, 1416, 1420,
1421, 1423, 1424, 1428, 1429, 1434, 1437, 1440, 1445,
1448, 1453, 1454, 1465, 1467, 1471, 1475, 1476, 1485,
1497, 1503, 1506, 1507, 1512, 1513, 1514, 1518, 1520,
1521, 1525, 1527, 1532, 1538, 1551, 1554, 1555, 1556,
1557, 1559, 1560, 1561, 1566, 1567, 1568, 1576, 1584,
1586, 1588, 1589, 1590, 1591, 1592, 1596, 1597, 1598,
1599, 1600, 1601, 1602, 1604, 1606, 1610, 1615, 1621,
1622, 1626, 1636, 1637, 1642, 1644, 1645, 1652, 1653,
1657, 1658, 1659, 1660, 1667, 1671, 1673, 1676, 1684,
1685, 1686, 1690, 1694, 1699, 1708, 1709, 1710, 1712,
1714, 1715, 1716, 1721, 1722, 1724, 1731, 1737, 1738,
1740, 1744, 1746, 1748, 1752, 1753, 1754, 1757, 1759,
1763, 1765, 1768, 1769, 1770, 1772, 1773, 1775, 1778,
1782, 1786, 1795, 1796, 1804, 1805, 1810, 1811, 1813,
1814, 1815, 1818, 1819, 1827, 1830, 1831, 1834, 1839,
1844, 1845, 1846, 1848, 1851, 1852, 1853, 1857, 1858,
1859, 1866, 1867, 1870, 1871, 1872, 1876, 1880, 1882,
1886, 1887, 1892, 1893, 1894, 1898, 1899, 1900, 1909,
1912, 1914, 1918, 1919, 1920, 1924, 1927, 1930, 1933,
1934, 1937, 1939, 1941, 1942, 1945, 1946, 1949, 1951,
1953, 1955, 1961, 1963, 1968, 1972, 1973, 1979, 1982,
1983, 1991, 1999, 2002, 2006, 2012, 2015, 2016, 2017,
2018, 2020, 2021, 2023, 2024, 2025, 2026, 2027, 2028,
2030, 2031, 2034, 2038, 2039, 2041, 2042, 2043, 2045,
2047, 2052, 2053, 2054, 2057, 2058, 2065, 2067, 2068,
2069, 2071, 2079, 2083, 2088, 2090, 2091, 2092, 2093,
2095, 2098, 2102, 2104, 2106, 2108, 2117, 2118, 2119,
2123, 2127, 2130, 2132, 2138, 2143, 2146, 2149, 2151,
2157, 2163, 2166, 2167, 2168, 2174, 2175, 2176, 2179,
2182, 2184, 2187, 2189, 2190, 2191, 2194, 2196, 2203,
2208, 2213, 2215, 2216, 2218, 2220, 2222, 2223, 2224,

```
2227,  2230,  2238,  2241,  2242,  2244,  2249,  2250,  2251,
2254,  2257,  2261,  2268,  2269,  2273,  2274,  2282,  2285,
2296,  2297,  2299,  2300,  2302,  2304,  2305,  2308,  2313,
2318,  2321,  2322,  2327,  2334,  2335,  2339,  2342,  2346,
2351,  2352,  2353,  2354,  2355,  2361,  2362,  2365,  2374,
2375,  2378,  2379,  2380,  2381,  2382,  2383,  2390,  2395,
2399,  2400,  2402,  2407,  2408,  2413,  2414,  2416,  2417,
2418,  2421,  2423,  2425,  2426,  2428,  2434,  2435,  2437,
2441,  2442,  2444,  2445,  2446,  2449,  2452,  2454,  2457,
2460,  2465,  2467,  2468,  2470,  2478,  2480,  2481,  2483,
2487,  2491,  2499,  2501

-------------------------------------------------------------------------------
START OF PASS NUMBER 3 OF 5 at Sun Dec  6 14:33:11 1998


Genetic Search Results
======================

*****  REACHED CONVERGENCE LIMIT:

   - generation number 23 at Sun Dec  6 14:34:11 1998
        - best fitness to date =     1.905
        - accuracy             =    67.332% (843/1252 correct)
        - correct              =   281 positive,   562 negative
        - incorrect            =   310 positive,    99 negative
        - average fitness      =     1.867
        - convergence factor   =     0.980
        - last improvement     = generation #16


-------------------------------------------------------------------------------
END OF PASS 3 at Sun Dec  6 14:34:11 1998 after 23 total generations

CPU usage for this pass =       20.120 seconds
```

```
Best individual found (at generation   16)
==========================================

 - fitness  =     1.90504

 - accuracy =    67.33227% (843/1252 training instances correct)
            =    65.49521% (820/1252 recall   instances correct)

 - length   =    88 bits =   2 rules at 44 bits/rule


 - unpruned rules
   --------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   0111 0011 1110 01 0001 11 0101111 11111111 11 1111 111

 - pruned rules
   ------------
   0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
   ==>  (Nodes=3-4 OR Nodes=>4)
   0111 0011 1110 01 0001 11 0101111 11111111 11 1111 111
   ==>  Nodes<>0 AND (Vessel Invasion=2 OR Vessel Invasion=3) AND
          Treatment<>CONCT AND Tumour Size=>20 AND Path Grade=3 AND
         (Age=31-40 OR Age=46-50 OR Age=51-55 OR Age=56-60 OR Age=>60)


 - confusion matrix:
                                    classified
                                    ----------

                               positive   negative

                   positive      255        326
      actual:
                   negative      106        565
```

- instances which were classified incorrectly:

1254, 1256, 1266, 1269, 1271, 1272, 1273, 1274, 1276,
1278, 1279, 1281, 1282, 1284, 1287, 1288, 1290, 1291,
1292, 1293, 1295, 1296, 1297, 1299, 1301, 1310, 1313,
1315, 1319, 1321, 1324, 1327, 1331, 1334, 1336, 1337,
1340, 1342, 1343, 1344, 1348, 1354, 1372, 1375, 1382,
1385, 1386, 1387, 1389, 1393, 1395, 1398, 1399, 1400,
1403, 1411, 1415, 1416, 1420, 1421, 1423, 1424, 1425,
1428, 1429, 1434, 1439, 1440, 1445, 1448, 1453, 1454,
1465, 1466, 1467, 1471, 1475, 1478, 1485, 1487, 1497,
1503, 1506, 1507, 1512, 1513, 1514, 1518, 1520, 1521,
1525, 1527, 1538, 1551, 1554, 1556, 1557, 1559, 1561,
1566, 1567, 1568, 1584, 1586, 1588, 1590, 1591, 1592,
1596, 1597, 1598, 1599, 1600, 1601, 1602, 1604, 1606,
1610, 1615, 1621, 1622, 1626, 1636, 1637, 1641, 1642,
1645, 1653, 1658, 1659, 1660, 1667, 1671, 1673, 1676,
1684, 1685, 1686, 1690, 1699, 1704, 1708, 1709, 1710,
1712, 1714, 1715, 1716, 1720, 1722, 1724, 1731, 1737,
1738, 1740, 1744, 1746, 1748, 1752, 1753, 1754, 1757,
1759, 1761, 1763, 1765, 1768, 1769, 1770, 1772, 1773,
1775, 1778, 1782, 1786, 1795, 1796, 1805, 1810, 1811,
1813, 1814, 1815, 1818, 1819, 1827, 1830, 1831, 1834,
1839, 1844, 1845, 1846, 1848, 1851, 1852, 1853, 1857,
1858, 1859, 1867, 1870, 1871, 1872, 1876, 1880, 1882,
1886, 1892, 1893, 1894, 1898, 1899, 1900, 1909, 1912,
1915, 1918, 1919, 1920, 1921, 1924, 1926, 1927, 1930,
1931, 1933, 1934, 1936, 1937, 1939, 1941, 1942, 1945,
1946, 1949, 1951, 1953, 1955, 1961, 1963, 1968, 1972,
1973, 1979, 1982, 1983, 1991, 2000, 2002, 2006, 2015,
2016, 2017, 2018, 2020, 2021, 2023, 2024, 2025, 2026,
2027, 2028, 2030, 2031, 2034, 2038, 2039, 2041, 2042,
2043, 2045, 2047, 2052, 2053, 2054, 2057, 2058, 2066,
2067, 2068, 2069, 2079, 2081, 2083, 2088, 2089, 2090,
2091, 2092, 2093, 2098, 2102, 2104, 2106, 2108, 2118,
2119, 2123, 2127, 2130, 2132, 2143, 2146, 2151, 2157,
2163, 2166, 2167, 2168, 2171, 2174, 2175, 2176, 2177,
2178, 2179, 2181, 2182, 2184, 2187, 2189, 2190, 2191,
2194, 2196, 2207, 2208, 2213, 2215, 2216, 2220, 2222,
2223, 2224, 2230, 2238, 2241, 2242, 2244, 2249, 2250,

```
2251,  2254,  2257,  2261,  2268,  2269,  2273,  2274,  2282,
2283,  2296,  2297,  2299,  2300,  2302,  2304,  2305,  2307,
2313,  2318,  2321,  2322,  2327,  2334,  2335,  2339,  2346,
2350,  2351,  2352,  2353,  2354,  2355,  2361,  2362,  2365,
2374,  2378,  2379,  2380,  2381,  2382,  2383,  2390,  2394,
2395,  2399,  2400,  2402,  2407,  2408,  2413,  2414,  2416,
2417,  2418,  2421,  2423,  2425,  2426,  2427,  2428,  2434,
2435,  2437,  2440,  2441,  2442,  2444,  2445,  2446,  2449,
2450,  2452,  2453,  2454,  2457,  2460,  2465,  2467,  2468,
2470,  2478,  2480,  2481,  2483,  2487,  2491,  2499,  2501

--------------------------------------------------------------------------------
START OF PASS NUMBER 4 OF 5 at Sun Dec  6 14:34:11 1998


Genetic Search Results
======================

*****  REACHED CONVERGENCE LIMIT:

   - generation number 18 at Sun Dec  6 14:34:57 1998
       - best fitness to date =     1.797
       - accuracy             =    66.374% (831/1252 correct)
       - correct              =   262 positive,   569 negative
       - incorrect            =   329 positive,    92 negative
       - average fitness      =     1.765
       - convergence factor   =     0.982
       - last improvement     = generation #0


--------------------------------------------------------------------------------
END OF PASS 4 at Sun Dec  6 14:34:57 1998 after 18 total generations

CPU usage for this pass =       15.090 seconds
```

```
Best individual found (at generation    0)
=======================================

- fitness  =     1.79738

- accuracy =    66.37380% (831/1252 training instances correct)
           =    64.29713% (805/1252 recall   instances correct)

- length   =    44 bits =   1 rule at 44 bits/rule


- unpruned rules
  --------------
  0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111

- pruned rules
  ------------
  0011 1111 1111 11 1111 11 1111111 11111111 11 1111 111
  ==>  (Nodes=3-4 OR Nodes=>4)


- confusion matrix:
                                  classified
                                  ----------

                             positive   negative

                 positive      233        348
     actual:
                 negative       99        572


- instances which were classified incorrectly:

        1254,  1256,  1266,  1269,  1271,  1272,  1273,  1274,  1276,
        1278,  1279,  1281,  1282,  1284,  1287,  1288,  1290,  1291,
        1292,  1293,  1295,  1296,  1297,  1299,  1301,  1310,  1313,
        1315,  1319,  1321,  1324,  1327,  1331,  1334,  1336,  1337,
        1340,  1342,  1343,  1344,  1348,  1354,  1372,  1375,  1382,
        1385,  1386,  1387,  1389,  1393,  1395,  1398,  1399,  1400,
```

1403, 1411, 1415, 1416, 1420, 1421, 1423, 1424, 1425,
1428, 1429, 1434, 1439, 1440, 1445, 1448, 1453, 1454,
1465, 1467, 1471, 1475, 1478, 1485, 1497, 1503, 1506,
1507, 1512, 1513, 1514, 1518, 1520, 1521, 1525, 1527,
1532, 1538, 1551, 1554, 1555, 1556, 1557, 1558, 1559,
1560, 1561, 1566, 1567, 1568, 1584, 1586, 1588, 1590,
1591, 1592, 1596, 1597, 1598, 1599, 1600, 1601, 1602,
1604, 1606, 1610, 1615, 1621, 1622, 1624, 1626, 1636,
1637, 1641, 1642, 1645, 1652, 1653, 1658, 1659, 1660,
1667, 1671, 1673, 1676, 1684, 1685, 1686, 1690, 1694,
1699, 1704, 1708, 1709, 1710, 1712, 1714, 1715, 1716,
1720, 1721, 1722, 1724, 1731, 1737, 1738, 1740, 1744,
1746, 1748, 1752, 1753, 1754, 1757, 1759, 1761, 1763,
1765, 1768, 1770, 1772, 1773, 1775, 1778, 1782, 1786,
1795, 1796, 1802, 1804, 1805, 1810, 1811, 1813, 1814,
1815, 1818, 1819, 1827, 1830, 1831, 1833, 1834, 1839,
1844, 1845, 1846, 1848, 1851, 1852, 1853, 1857, 1858,
1859, 1867, 1870, 1871, 1872, 1876, 1880, 1882, 1886,
1892, 1893, 1894, 1898, 1899, 1900, 1909, 1912, 1914,
1915, 1918, 1919, 1920, 1921, 1924, 1926, 1927, 1930,
1933, 1934, 1936, 1937, 1939, 1941, 1942, 1945, 1946,
1949, 1951, 1953, 1955, 1961, 1963, 1968, 1972, 1973,
1979, 1982, 1983, 1999, 2000, 2002, 2006, 2015, 2016,
2017, 2018, 2020, 2021, 2023, 2024, 2025, 2026, 2027,
2028, 2030, 2031, 2034, 2038, 2039, 2041, 2042, 2043,
2045, 2047, 2052, 2053, 2054, 2057, 2058, 2061, 2066,
2067, 2068, 2069, 2071, 2079, 2081, 2083, 2088, 2089,
2090, 2091, 2092, 2093, 2098, 2102, 2104, 2106, 2108,
2117, 2118, 2119, 2123, 2127, 2130, 2132, 2138, 2143,
2146, 2149, 2151, 2157, 2163, 2166, 2167, 2168, 2171,
2174, 2175, 2176, 2177, 2179, 2181, 2182, 2184, 2187,
2189, 2190, 2191, 2192, 2194, 2196, 2207, 2208, 2213,
2215, 2216, 2220, 2222, 2223, 2224, 2230, 2238, 2241,
2242, 2244, 2249, 2250, 2251, 2254, 2257, 2261, 2268,
2269, 2273, 2274, 2282, 2283, 2290, 2296, 2297, 2299,
2300, 2302, 2304, 2305, 2307, 2313, 2314, 2318, 2321,
2322, 2327, 2334, 2335, 2339, 2342, 2346, 2350, 2351,
2352, 2353, 2354, 2355, 2361, 2362, 2365, 2374, 2378,
2379, 2380, 2381, 2382, 2383, 2390, 2394, 2395, 2399,
2400, 2402, 2407, 2408, 2413, 2414, 2416, 2417, 2418,

```
        2421,  2423,  2426,  2427,  2428,  2434,  2435,  2437,  2440,
        2441,  2442,  2444,  2445,  2446,  2449,  2450,  2452,  2453,
        2454,  2457,  2460,  2465,  2467,  2468,  2470,  2478,  2480,
        2481,  2483,  2487,  2491,  2499,  2501

--------------------------------------------------------------------------------
START OF PASS NUMBER 5 OF 5 at Sun Dec  6 14:34:57 1998


Genetic Search Results
======================

*****  REACHED CONVERGENCE LIMIT:

   - generation number 40 at Sun Dec  6 14:36:42 1998
        - best fitness to date =     2.087
        - accuracy             =    68.930% (863/1252 correct)
        - correct              =   321 positive,   542 negative
        - incorrect            =   270 positive,   119 negative
        - average fitness      =     2.046
        - convergence factor   =     0.980
        - last improvement     = generation #38


--------------------------------------------------------------------------------
END OF PASS 5 at Sun Dec  6 14:36:42 1998 after 40 total generations

CPU usage for this pass =       34.480 seconds


Best individual found (at generation   38)
==========================================

 - fitness  =     2.08655

 - accuracy =    68.92971% (863/1252 training instances correct)
            =    64.21725% (804/1252 recall   instances correct)

 - length   =    88 bits =   2 rules at 44 bits/rule
```

- unpruned rules

```
0011 1111 1111 11 1011 11 0111111 11111111 11 1111 111
0101 1111 1111 01 1111 11 1111011 11111111 11 1101 111
```

- pruned rules

```
0011 1111 1111 11 1011 11 0111111 11111111 11 1111 111
==>  (Nodes=3-4 OR Nodes=>4) AND Path Grade<>1 AND Age<><31
0101 1111 1111 01 1111 11 1111011 11111111 11 1101 111
==>  (Nodes=1-2 OR Nodes=>4) AND Tumour Size=>20 AND Age<>51-55 AND
      Amenorrhea rec<>3
```

- confusion matrix:

		classified	
		positive	negative
actual:	positive	290	291
	negative	157	514

- instances which were classified incorrectly:

1254, 1256, 1258, 1260, 1262, 1263, 1266, 1267, 1269,
1271, 1272, 1273, 1274, 1276, 1278, 1281, 1282, 1284,
1287, 1288, 1290, 1291, 1292, 1293, 1295, 1297, 1298,
1299, 1301, 1303, 1310, 1313, 1315, 1319, 1327, 1331,
1334, 1336, 1340, 1342, 1343, 1344, 1347, 1350, 1352,
1355, 1375, 1379, 1382, 1385, 1386, 1387, 1390, 1395,
1397, 1398, 1399, 1400, 1403, 1406, 1411, 1415, 1416,
1417, 1420, 1421, 1424, 1425, 1428, 1429, 1434, 1437,
1439, 1440, 1445, 1448, 1453, 1454, 1465, 1467, 1471,
1475, 1478, 1484, 1485, 1492, 1493, 1497, 1503, 1506,
1507, 1512, 1513, 1514, 1518, 1520, 1525, 1527, 1537,

1538, 1545, 1547, 1551, 1554, 1555, 1556, 1557, 1559,
1561, 1564, 1566, 1567, 1578, 1584, 1586, 1588, 1590,
1591, 1592, 1596, 1597, 1598, 1599, 1600, 1602, 1604,
1606, 1608, 1610, 1613, 1615, 1616, 1621, 1622, 1626,
1627, 1636, 1637, 1639, 1640, 1641, 1642, 1645, 1653,
1658, 1659, 1660, 1663, 1667, 1669, 1671, 1673, 1676,
1684, 1685, 1688, 1690, 1694, 1698, 1699, 1704, 1707,
1708, 1709, 1710, 1712, 1715, 1716, 1722, 1724, 1729,
1731, 1732, 1737, 1738, 1744, 1746, 1748, 1753, 1756,
1757, 1759, 1761, 1762, 1763, 1765, 1768, 1770, 1772,
1773, 1778, 1782, 1786, 1793, 1795, 1796, 1802, 1804,
1805, 1809, 1810, 1811, 1813, 1814, 1815, 1818, 1819,
1827, 1830, 1831, 1833, 1834, 1839, 1844, 1846, 1848,
1851, 1852, 1853, 1857, 1859, 1866, 1867, 1871, 1872,
1875, 1876, 1880, 1886, 1887, 1890, 1892, 1893, 1894,
1895, 1899, 1900, 1909, 1912, 1914, 1918, 1919, 1921,
1927, 1930, 1933, 1934, 1937, 1939, 1941, 1942, 1945,
1946, 1948, 1949, 1951, 1953, 1955, 1961, 1968, 1972,
1973, 1979, 1982, 1983, 1991, 1993, 1996, 2002, 2010,
2012, 2015, 2016, 2017, 2018, 2020, 2021, 2023, 2024,
2025, 2026, 2028, 2029, 2030, 2031, 2034, 2038, 2039,
2041, 2042, 2045, 2047, 2052, 2053, 2054, 2057, 2058,
2065, 2067, 2068, 2069, 2071, 2079, 2083, 2088, 2091,
2092, 2093, 2095, 2099, 2101, 2102, 2104, 2106, 2108,
2119, 2123, 2127, 2130, 2132, 2133, 2143, 2146, 2150,
2151, 2157, 2163, 2165, 2166, 2167, 2171, 2173, 2174,
2175, 2176, 2177, 2178, 2179, 2181, 2184, 2187, 2189,
2190, 2191, 2192, 2194, 2199, 2203, 2207, 2208, 2213,
2216, 2220, 2222, 2223, 2224, 2230, 2241, 2242, 2244,
2247, 2248, 2249, 2250, 2251, 2254, 2257, 2261, 2265,
2268, 2269, 2273, 2274, 2282, 2285, 2291, 2296, 2297,
2299, 2300, 2302, 2304, 2305, 2307, 2313, 2316, 2317,
2318, 2321, 2322, 2335, 2339, 2346, 2351, 2353, 2354,
2355, 2361, 2365, 2374, 2378, 2380, 2382, 2383, 2390,
2394, 2395, 2399, 2400, 2402, 2407, 2413, 2414, 2417,
2418, 2421, 2423, 2424, 2426, 2429, 2434, 2437, 2440,
2441, 2442, 2444, 2445, 2446, 2449, 2450, 2451, 2452,
2454, 2457, 2460, 2462, 2465, 2468, 2470, 2474, 2475,
2478, 2479, 2481, 2483, 2487, 2491, 2499

END OF RUN at Sun Dec 6 14:36:42 1998

Total CPU usage = 109.730 seconds

Voting Results from All Passes

- accuracy on recall instances = 64.217252% (804/1252)

- recall instances considered unclassifiable = 0 of 1252

- confusion matrix:

		classified	
		positive	negative
actual:	positive	290	291
	negative	157	514

- instances which were classified incorrectly:

1254, 1256, 1258, 1260, 1262, 1263, 1266, 1267, 1269,
1271, 1272, 1273, 1274, 1276, 1278, 1281, 1282, 1284,
1287, 1288, 1290, 1291, 1292, 1293, 1295, 1297, 1298,
1299, 1301, 1303, 1310, 1313, 1315, 1319, 1327, 1331,
1334, 1336, 1340, 1342, 1343, 1344, 1347, 1350, 1352,
1355, 1375, 1379, 1382, 1385, 1386, 1387, 1390, 1395,
1397, 1398, 1399, 1400, 1403, 1406, 1411, 1415, 1416,
1417, 1420, 1421, 1424, 1425, 1428, 1429, 1434, 1437,
1439, 1440, 1445, 1448, 1453, 1454, 1465, 1467, 1471,
1475, 1478, 1484, 1485, 1492, 1493, 1497, 1503, 1506,
1507, 1512, 1513, 1514, 1518, 1520, 1525, 1527, 1537,
1538, 1545, 1547, 1551, 1554, 1555, 1556, 1557, 1559,

1561, 1564, 1566, 1567, 1578, 1584, 1586, 1588, 1590,
1591, 1592, 1596, 1597, 1598, 1599, 1600, 1602, 1604,
1606, 1608, 1610, 1613, 1615, 1616, 1621, 1622, 1626,
1627, 1636, 1637, 1639, 1640, 1641, 1642, 1645, 1653,
1658, 1659, 1660, 1663, 1667, 1669, 1671, 1673, 1676,
1684, 1685, 1688, 1690, 1694, 1698, 1699, 1704, 1707,
1708, 1709, 1710, 1712, 1715, 1716, 1722, 1724, 1729,
1731, 1732, 1737, 1738, 1744, 1746, 1748, 1753, 1756,
1757, 1759, 1761, 1762, 1763, 1765, 1768, 1770, 1772,
1773, 1778, 1782, 1786, 1793, 1795, 1796, 1802, 1804,
1805, 1809, 1810, 1811, 1813, 1814, 1815, 1818, 1819,
1827, 1830, 1831, 1833, 1834, 1839, 1844, 1846, 1848,
1851, 1852, 1853, 1857, 1859, 1866, 1867, 1871, 1872,
1875, 1876, 1880, 1886, 1887, 1890, 1892, 1893, 1894,
1895, 1899, 1900, 1909, 1912, 1914, 1918, 1919, 1921,
1927, 1930, 1933, 1934, 1937, 1939, 1941, 1942, 1945,
1946, 1948, 1949, 1951, 1953, 1955, 1961, 1968, 1972,
1973, 1979, 1982, 1983, 1991, 1993, 1996, 2002, 2010,
2012, 2015, 2016, 2017, 2018, 2020, 2021, 2023, 2024,
2025, 2026, 2028, 2029, 2030, 2031, 2034, 2038, 2039,
2041, 2042, 2045, 2047, 2052, 2053, 2054, 2057, 2058,
2065, 2067, 2068, 2069, 2071, 2079, 2083, 2088, 2091,
2092, 2093, 2095, 2099, 2101, 2102, 2104, 2106, 2108,
2119, 2123, 2127, 2130, 2132, 2133, 2143, 2146, 2150,
2151, 2157, 2163, 2165, 2166, 2167, 2171, 2173, 2174,
2175, 2176, 2177, 2178, 2179, 2181, 2184, 2187, 2189,
2190, 2191, 2192, 2194, 2199, 2203, 2207, 2208, 2213,
2216, 2220, 2222, 2223, 2224, 2230, 2241, 2242, 2244,
2247, 2248, 2249, 2250, 2251, 2254, 2257, 2261, 2265,
2268, 2269, 2273, 2274, 2282, 2285, 2291, 2296, 2297,
2299, 2300, 2302, 2304, 2305, 2307, 2313, 2316, 2317,
2318, 2321, 2322, 2335, 2339, 2346, 2351, 2353, 2354,
2355, 2361, 2365, 2374, 2378, 2380, 2382, 2383, 2390,
2394, 2395, 2399, 2400, 2402, 2407, 2413, 2414, 2417,
2418, 2421, 2423, 2424, 2426, 2429, 2434, 2437, 2440,
2441, 2442, 2444, 2445, 2446, 2449, 2450, 2451, 2452,
2454, 2457, 2460, 2462, 2465, 2468, 2470, 2474, 2475,
2478, 2479, 2481, 2483, 2487, 2491, 2499

Per-pass statistics
==========

- average fitness = 1.91964

- average accuracy = 67.46006 (845/1252)
= 64.64857 (809/1252)

- average CPU time = 21.80400

- average generation count = 24