MEDICAL DATA MINING USING EVOLUTIONARY COMPUTATION

By

Ngan Po Shun

A Dissertation

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Department of Computer Science and Engineering

The Chinese University of Hong Kong

June 1998

Abstract

Data mining is an automated process of discovering knowledge from databases. There are various kinds of data mining methods aiming to search for different kind of knowledge. In this thesis, data mining approaches that use rules and Bayesian network as the knowledge representations are described. Rule can represent interesting patterns and regularities in the database, while a Bayesian network can represent the overall structure of the relationships among the attributes. We investigate the use of Evolutionary Computation as the search algorithm for data mining. Evolutionary Computation is a kind of weak search methods that simulates the natural evolution. It is a general search technique and does not require any domain specific techniques.

We present an approach for rule learning that uses Generic Genetic Programming as the core search technique. It is a grammar based search technique that allows a powerful knowledge representation. The grammar serves as a template to specify the format of rules. A technique called token competition is employed to learn multiple rules from the data.

In learning Bayesian networks from data, a problem is that a Bayesian network can only be constructed from discrete variables. We investigate the use of genetic algorithm for learning a policy for discretization of continuous variables while learning the Bayesian network structure. The experiments show that this approach performs better than the greedy approach.

A system for knowledge discovery that combines the approaches of rule learning and Bayesian network learning is developed. We introduce the applications of the system to two real-life medical databases for limb fracture and Scoliosis. The

knowledge discovered provides insights to the clinicians and allows them to have a better understanding of these two medical domains.

摘要

數據勘探 (Data Mining) 是指從數據庫中自動地發現知識。 這論文會詳論兩種搜索方法， 分別從數據庫中發掘規則 (Rule) 及貝氏網絡 (Bayesian Network)， 用以表達當中的知識。 規則能表達數據中的規律， 貝氏網絡則可表達數據庫中屬性之間的關系構造。 而我們使用了進化計算 (Evolutionary Computation) 作爲搜索算法。

在學習規則中， 我們使用了全面遺傳程序 （Generic Genetic Programming）。 其中的文法 (grammar) 能規定規則的格式。 另外， 我們使用了一個名爲資源競爭 (token competition) 的技巧，用以發掘多條的規則。 在學習貝氏網絡中， 其一問題爲如何處理連續變數 (continuous variables)。 我們研究了使用遺傳算法 (Genetic Algorithm)， 將連續變數分段 (discretization)，以及同時學習貝氏網絡的結構。 實驗顯示這個方法的結果比貪婪方法好。

我們並會介紹一個以上述兩種方法爲基礎的數據勘探系統。我們使用了這個系統分析兩個醫學上的數據庫， 發現了一些有意義的知識， 以增加醫學人員對這兩方面的了解。

Acknowledgements

I would like to express my deepest gratitude and appreciation to my supervisors Prof. K. S. Leung and Dr. M. L. Wong for their guidance on my research. They provided fruitful discussions and invaluable suggestions to me. I would like to thanks Dr. W. Lam and Dr. Ada Fu for their precious comments on the subject of this thesis. Moreover, I would like to thanks Prof. Jack Cheng for his kindly helps and explanations on the medical knowledge. I would also like to acknowledge Ms. Bella C. S. Lau and Ms. Vivian K. S. Lee for their supports and helps on building the data mining system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Databases are valuable treasures. A database not only stores and provides data but also contains hidden precious knowledge, which can be very important. It can be a new law in science, a new insight for curing a disease or a new market trend that can make millions of dollars. Conventionally, the data is analyzed manually. Many hidden and potentially useful relationships may not be recognized by the analyst. Nowadays, many organizations are capable of generating and collecting a huge amount of data. The size of data available now is beyond the capability of our mind to analyze. It requires the power of computers to handle it. Data mining, or knowledge discovery in database, is the automated process of sifting the data to get the gold buried in the database.

In this chapter, Section 1.1 is a brief introduction of the definition and the objectives of data mining. Section 1.2 states the research motivation. Section 1.3 lists the contributions of this thesis. The organization of this thesis is sketched in Section 1.4.

1.1 Data Mining

The two terms Data Mining and Knowledge Discovery in Database have similar meanings. The term *Data Mining* is commonly used by statisticians, to denote the finding of useful patterns in data. It consists of applying data analysis and

discovery algorithms to produce patterns or models over the data. On the other hand, *Knowledge Discovery in Database* (KDD) can be defined as the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (Fayyad et al. [1996]). The data are records in a database. The knowledge discovered from the KDD process should be unable to be obtained by a straightforward computation. The knowledge should be not yet discovered and should be beneficial to the user. The knowledge should be able to apply to new data with some degree of certainty. Finally the knowledge should be human understandable.

KDD is an interactive and iterative process comprises with several steps. In Fayyad et al. [1996], KDD is divided into several steps. Data Mining can be considered as one of the steps in the KDD process. Data mining is the core of the KDD process, and thus the two terms are often used interchangeably. The whole process of KDD can consist of five steps:

1. Selection is made to extract relevant or target data set from the database.

2. Preprocessing is needed to remove the noise and to handle missing data fields.

3. Transformation is performed to reduce the number of variables under consideration.

4. A suitable data mining algorithm is employed on the prepared data.

5. Finally the result of data mining is interpreted and evaluated.

If the discovered knowledge is not satisfactory, these steps will be iterated. The discovered knowledge can then be applied in decision making.

Different data mining algorithms aim to find different kinds of knowledge. Chen et al. [1996] grouped the techniques for knowledge discovery into six categories.

1. Mining of association rules finds rules in the form of "$A_1 \wedge \cdots \wedge A_m \Rightarrow B_1 \wedge \cdots \wedge B_n$", where A_i and B_j are attributes values. This association rule

tries to capture the association between the attributes. The rule means that if A_1 and $\cdots$ and A_m appear in a record, then B_1 and $\cdots$ and B_n will usually appear.

2. Data generalization and summarization summarized the general characteristics of a group of target class and presents the data in a high-level view.

3. Classification formulates a classification model based on the data. The model can be used to classify an unseen data item into one of the predefined classes based on the attribute values.

4. Data clustering identifies a finite set of clusters or categories to describe the data. Similar data items are grouped into a cluster such that the interclass similarity is maximized and the interclass similarity is minimized. The common characteristic of the cluster is analyzed and presented.

5. Pattern based similarity search tries to search for a pattern in temporal or spatial-temporal data, such as financial databases or multimedia databases.

6. Mining path traversal patterns tries to capture user access patterns in an information providing system, such as World Wide Web.

Machine learning (Carbonell et al. [1983]) and data mining share a similar objective. Machine learning learns a computer model from a set of training examples. Many machine learning algorithms can be applied to databases. Rather than learning on a set of instances, machine learning is done on a file of records from a database (Frawley et al. [1992]). However, databases are designed to meet the needs of real world applications. They are often dynamic, incomplete, noisy and much larger than typical machine learning data sets. These issues cause difficulties in direct application of machine learning methods.

1.2 Motivation

Data mining has recently become a popular research topic. The increasing use of computer results in an explosion of information. These data can be best used if the knowledge hidden inside can be uncovered. Thus there is a need for a way to automatically discover knowledge from data. The research in this area can be useful to a lot of real world problems. For instance, medical domain is a major area for applying data mining. With the computerization in hospitals, a huge amount of data has been collected. It is beneficial if these data can be analyzed automatically.

Learning from examples is not a new area in computer science. Machine learning has a well-developed history. Many classification approaches have been designed to construct a model for classification from a set of training cases. However, the goal of data mining is different from classification. The objective of data mining is not to classify all the unseen cases perfectly, but discover knowledge interesting to the users, even though the accuracy may be not high. Accuracy should be one of the requirements for an interesting knowledge, but an approach that can only find knowledge with high accuracy should not be a complete approach for data mining. In many real-life situations, strong rules just do not exist, or have already been discovered since the relationship is so obvious. It is important if the data mining method can discover weak rules.

Another requirement in data mining is that the knowledge discovered should be understandable by the user, such that the user can make decision based on the new knowledge. Some approaches are black box approaches and not suitable for data mining. Some approaches can give human understandable results, but the results may be just complicated and difficult to interpret. Rule is commonly used by human to represent knowledge, and should be a suitable knowledge representation in data mining. However, there are different representations of rule, with different representation power. Many rule learning approaches learn rules with its own format, and the format may not be powerful enough to represent the knowledge

hidden in the data. Moreover, the rule format may not be the one that the user desires. It is advantageous if the knowledge representation can be improved.

Another interesting knowledge representation is Bayesian network, which is based on a well-developed Bayesian probability theory. It is easy to understand because of its graphical representation. It can represent the overall causality between variables in the domain. One difficulty in Bayesian network learning is on how to handle the continuous variables. Friedman and Goldszmidt [1996] has proposed a measure for discretization of continuous variables. It is worthwhile to investigate the use of other search methods other than the greedy method they proposed.

Evolutionary computation is a kind of weak search method for optimization. It is a domain independent search method that can be applied to a wide range of problem. Algorithms in evolutionary computation can be used as a search method for knowledge discovery. It is suitable for hard search problems where domain specific techniques are not available or difficult to design. In this thesis, we will investigate the use of evolutionary computation to rule learning and Bayesian network learning, and the applications of these techniques on analyses of medical databases.

1.3 Contributions of the research

The contributions of the research are listed below, in the order that they appear in the thesis:

- An approach for rule learning have been developed. This approach uses Generic Genetic Programming (GGP) as the learning algorithm. We have designed a suitable grammar to represent a rule, and we have investigated how the grammar can be modified in order to learn rules with different formats. Other techniques have been employed in GGP to facilitate the learning: seeds are used to generate better rules, and the operator 'dropping condition' is used to generalize rules. The evaluation function is designed to

measure both the accuracy and significance of the rule, so that interesting rules can be learned.

- The technique token competition has been employed to learn multiple rules simultaneously. This technique effectively maintains groups of individuals in the population, with different groups evolving different rules.

- We have investigated the use of Genetic Algorithm in the process of discretizing continuous variables while learning Bayesian networks. A system has been implemented that alternatively learns a Bayesian network structure and a discretization policy from the data. The approach MDLEP is employed to learn the network structure. Genetic Algorithm is used to learn the discretization policy, and the performance has been compared with a greedy approach.

- We have developed a data mining system that consists of a causality analysis step and a rule learning step. These two steps are not independent processes. The Bayesian network discovered from the causality analysis can help the user to understand the domain, so that the user can construct a suitable grammar to guide rule learning, and the search space can be greatly decreased.

- We have applied the data mining system to two real-life medical databases. We have consulted the domain experts to understand the domains, so as to pre-process the data and construct suitable grammars for rule learning. The learning results have been fed back to the domain experts. Interesting knowledge are discovered, which can help the clinician to get a deeper understanding of the domains.

1.4 Organization of the thesis

Chapter 2 of this thesis in a literature review on different approaches of data mining. The approaches are grouped into decision tree approach, classification

rule learning, association rule mining, statistical approach and Bayesian network learning. Representative algorithms in each group will be introduced.

In chapter 3, we will introduce what is Evolutionary Computation, and describe four evolutionary algorithms: Genetic Algorithm, Genetic Programming, Evolutionary Programming and Evolution Strategy, as well as Generic Genetic Programming (GGP), which is an extension of Genetic Programming.

Chapter 4 and chapter 5 will discuss how evolutionary computation can be applied to discover rules from databases. Chapter 4 will focus on how the problem of rule learning is modeled such that GGP can be applied as the learning algorithm. The representation of rules, the genetic operators for evolving new rules, and the evaluation function will be introduced in this chapter. However, learning one rule from data is inadequate. Chapter 5 will describe how to learn multiple number of rules. The technique token competition is employed to solve this problem. A rule learning system will be introduced, and the experiment results on two machine learning databases will be presented in this chapter. The material of these two chapters have been published in (Ngan et al. [1998b]).

Chapter 6 will describe another problem: Bayesian network learning. We will first describe an approach, MDLEP, which learns Bayesian Network based on the Minimum Description Length (MDL) principle and Evolutionary Programming. The research on discretization of continuous variables in Bayesian Network learning based on MDL (Friedman and Goldszmidt [1996]) has been extended. Genetic Algorithm is used as the optimization method instead of the proposed greedy approach, and the experimental results will be presented.

In chapter 7, a system for data mining will be introduced. This system combines the approaches for Bayesian network learning and rule learning. The system has been used to analyze real-life medical databases for limb fracture and Scoliosis. The applications of this system and the learning results will be presented in this chapter. A paper on this system has been accepted for publication (Ngan et al. [1998a]).

Chapter 8 is a conclusion of this thesis. The research work will be summarized,

and some suggestions for future researches will be given.

Chapter 2

Related Work in Data Mining

There are a large variety of data mining approaches, with different search methods aiming to find different kinds of knowledge. This chapter reviews data mining approaches related to this research. Decision tree approach, classification rule learning, association rule mining, statistical approach and Bayesian network learning are reviewed in the following sections.

2.1 Decision Tree Approach

Decision Tree is a tree like structure that represents the knowledge for classification. Internal nodes in a decision tree are labeled with attributes, the edges are labeled with attribute values and the leaves are labeled with classes. An example of a decision tree is shown in Figure 2.1. This tree is for classifying whether the weather of a Saturday morning is good or not. It can classify the weather into the class P (positive) or N (negative). For a given record, the classification process starts on the root node. The attribute in the node is tested, and the value determines which edge is taken. This process is repeated until a leaf is reached. The record is then classified as the class of the leaf. Decision tree is a simple knowledge representation for representing a classification model, but the tree can be very complicate that is difficult to interpret.

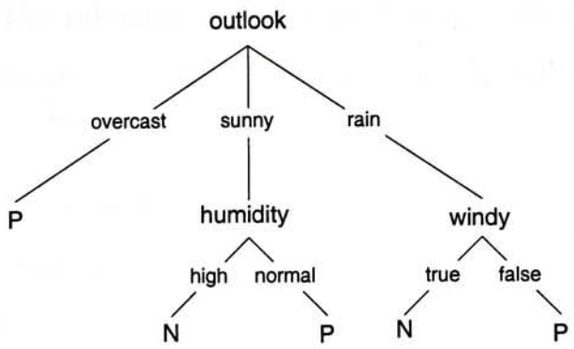


Figure 2.1: A decision tree

2.1.1 ID3

ID3 (Quinlan [1986]) is a simple algorithm to construct a decision tree from a set of training objects. It is a heuristic top-down irrevocable search. Initially the tree contains only a root node and all the training cases are placed in the root node. ID3 uses information as a criterion for selecting the branching attribute of a node. Let the node contains a set T of cases, with $|C_j|$ of the cases belonging to one of the pre-defined class C_j. The information needed for classification in the current node is

$$info(T) = -\sum_j \frac{|C_j|}{|T|} \log_2 \frac{|C_j|}{|T|} \tag{2.1}$$

This value measures the average amount of information needed to identify the class of a case. Assume that using attribute X as the branching attribute will divide the cases into n subsets. Let T_i denotes the set of cases in subset i. The information required for the subset i is $info(T_i)$. Thus the expected information required *after* choosing attribute X as the branching attribute is the weighted average of subtree information :

$$info_X(T) = \sum_i \frac{|T_i|}{|T|} \times info(T_i) \tag{2.2}$$

Thus the information gain will be

$$gain(X) = info(T) - info_X(T) \tag{2.3}$$

As a smaller value in the information corresponds to a easier classification, the attribute X with the maximum information gain is selected for the branching of the node.

After the branching attribute is selected, the training cases are divided by the different values of the branching attribute. If all examples in one branch belong to the same class, then this branch becomes a leaf labeled with that class. If all branches are labeled with a class, the algorithm terminates. Otherwise the process is recursively applied on each branch.

ID3 uses the chi-square test to avoid over-fitting to the noise. In a set T of cases, let o_{C_j,x_i} denote the number of records in class C_j with $X = x_i$. If attribute X is irrelevant for classification, the expected number of cases belonging to class C_j with $X = x_i$ is

$$e_{C_j,x_i} = |C_j| \times \frac{|T_i|}{|T|} \tag{2.4}$$

The value of chi-square is approximately

$$\chi^2 \approx \sum_i \sum_j \frac{(o_{C_j,x_i} - e_{C_j,x_i})^2}{e_{C_j,x_i}} \tag{2.5}$$

In choosing the branching attribute for the decision tree, if χ^2 is lower than a threshold, then that attribute will not be used. This can avoid creating unnecessary branches that complicate the constructed tree .

2.1.2 C4.5

C4.5 (Quinlan [1993]) is the successor of ID3. The use of information gain in ID3 has a serious deficiency that it favors tests with many outcomes. C4.5 improves this by using a gain ratio as the criterion for selecting the branching attribute. A value *split* $info(X)$ is defined with a similar definition of $info(X)$

$$split\ info(X) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \log_2 \frac{|T_i|}{|T|} \tag{2.6}$$

This value represents the potential information generated by dividing T into n subsets. The gain ratio is used as the new criterion

$$gain\ ratio(X) = gain(X)/split\ info(X) \tag{2.7}$$

The attribute with the maximum value on $gain\ ratio(X)$ is selected as the branching attribute.

C4.5 abandoned the chi-square test for avoiding over-fitting. Rather, C4.5 allows the tree to grow and later prunes the unnecessary branches. The tree pruning step replaces a subtree by a leaf or the most frequently used branch. The decision on whether a subtree is pruned depends on an estimation of the error rate. Suppose that a leaf gives an error of E out of N training cases. For a given confidence level CF, the upper limit of the error probability for the binomial distribution is written as $U_{CF}(E, N)$. The upper limit is used as the pessimistic error rate of the leaf. The estimated number of errors for a leaf covering N training cases is thus $N \times U_{CF}(E, N)$. The estimated number of errors for a subtree is the sum of errors of its branches.

Pruning is performed if replacing a subtree by a leaf or a branch can give a lower estimated number of errors. For example, for a subtree with three leaves, which respectively covers 6, 9 and 1 training cases without errors, the estimated number of mis-classification with the default confidence level of 25% is

$$6\times U_{25\%}(0,6)+9\times U_{25\%}(0,9)+1\times U_{25\%}(0,1) = 6\times 0.206+9\times 0.143+1\times 0.750 = 3.273.$$

If they are combined to a leaf node, it mis-classifies 1 out of 16 training case. The estimated number of mis-classifications of this leaf is

$$16 \times U_{25\%}(1, 16) = 16 \times 0.157 = 2.512.$$

This is better than the original subtree and thus the leaf can replace the original subtree.

2.2 Classification Rule Learning

A rule is a sentence of the form "if *antecedents*, then *consequent*". Rules are commonly used in expressing knowledge and are easily understood by human. Rules are also commonly used in expert systems for decision making. Rule learning is the process of inducing rules from a set of training examples. Many algorithms in rule learning try to search for rules to classify a case into one of the pre-specified classes.

2.2.1 AQ algorithm

AQ (Michalski [1969]) is a family of algorithms for inductive learning. One example is AQ15 (Michalski et al. [1986]). The knowledge representation used in AQ is the decision rule. A rule is represented in Variable-valued Logic system 1 (VL_1). In VL_1, a *selector* relates a variable to a value or a disjunction of values, e.g. color = red $\vee$ green. A conjunction of selectors forms a *complex*. A *cover* is a disjunction of complexes describing all positive examples and none of the negative examples. A cover defines the antecedents of a decision rule. The original AQ can only construct exact rules, i.e. for each class, the decision rule must cover only the positive examples and none of the negative examples.

AQ algorithm is a covering method instead of the divide-and-conquer method of ID3. The search algorithm can be described as follows (Michalski [1983]):

1. A positive example, called the *seed*, is chosen from the training examples.

2. A set of complexes, called a *star*, that covers the seed is generated by the star generating step. Each complex in the star must be the most general without covering a negative example.

3. The complexes in the star is ordered by the lexicographic evaluation function (LEF). A commonly used LEF is to maximize the number of positive examples covered.

4. The examples covered by the best complex is removed from the training examples

5. The best complex in the star is added to the cover.

6. Steps 1-5 are repeated until the cover can cover all the positive examples.

The searching in the star generating step (step 2) is a top down irrevocable beam search. This step can be summarized as follows:

1. Let the partial star be the set containing the empty complex, i.e. without any selector.

2. While the partial star covers negative examples,

   (a) Select a covered negative example.

   (b) Let *extension* be the set of all selectors that cover the seed but not the negative example.

   (c) Update the partial star to be the set $\{x \wedge y \mid x \in \text{partial star}, y \in extension\}$.

   (d) Remove all complexes in the partial star subsumed by other complexes.

3. Trim the partial star, i.e. retain only the *maxstar* best complexes.

In the star generating step, not all the complexes that cover the seed are included. The partial star will be trimmed by retaining only *maxstar* best complexes. The heuristic used is to retain the complexes that "maximize the sum of positive examples covered and negative examples excluded".

2.2.2 CN2

CN2 (Clark and Niblett [1989]) incorporates ideas from both AQ and ID3 algorithm. AQ algorithm cannot handle noise properly. CN2 retains the beam search of AQ algorithm but removed its dependence on specific training examples (the

seeds) during the search. CN2 uses a decision list as the knowledge representation. A decision list is a list of pairs (ϕ_1, C_1), (ϕ_2, C_2), ..., (ϕ_r, C_r),where ϕ_i is a complex, C_i is a class, and the last description ϕ_r is the constant true. This list means ' if $\phi_1$then C_1 else if ϕ_2 then $C_2 \ldots$.else C_r'.

Each step of CN2 searches for a complex that covers a large number of examples of class C and a small number of other classes. Having found a good complex, the algorithm removes those examples it covers from the training set and adds the rule 'if <complex> then predict C' to the end of rule list. This step is repeated until no more satisfactory complexes can be found.

The searching algorithm for a good complex is a beam search. At each stage in the search, CN2 stores a *star* S of 'set of best complexes found so far'. The star is initialized to the empty complex. The complexes of the star are then specialized by intersecting with all possible selectors. Each specialization is similar to introducing a new branch in ID3. All specializations of complexes in the star are examined and ordered by the evaluation criteria. Then the star is trimmed to size *maxstar* by removing the worst complexes. This process of searching is iterated until no further complexes that exceed the threshold of evaluation criteria can be generated.

The evaluation criteria for complexes consist of two tests for testing the prediction accuracy and significance of the complex. Let $(p_1, \ldots, p_n)$ be the probability of examples in class $C_1, \ldots C_n$. CN2 uses the information theoretic entropy (Equation 2.1)

$$info = -\sum_i p_i \log_2(p_i) \tag{2.8}$$

to measure the quality of complex (lower the entropy, the better the quality). The likelihood ratio statistic is used to measure the significance of complex :

$$2\sum_{i=1}^{n} f_i \log(f_i/e_i) \tag{2.9}$$

where $(f_1, \ldots, f_n)$ is the observed frequency distribution and $(e_1, \ldots, e_n)$ is the expected distribution. A complex with a high value of this ratio means the high

accuracy on training data is not just due to chance.

2.2.3 C4.5RULES

Other than being able to produce a decision tree as described in section 2.1.2, a component of C4.5, C4.5RULES (Quinlan [1993]), can transform the constructed decision tree by C4.5 into production rules. Each path of the decision tree from the root to the leaf equals to a rule. The antecedent of the rule contains all the conditions of the path, and the consequent is the class of the leaf. However this rule can be very complicate and a simplification is required. Suppose that the rule gives E errors out of the N covered cases, and if condition X is removed from the rule, the rule will give E_{x^-} errors out of the N_{x^-} covered cases. If the pessimistic error $U_{CF}(E_{x^-}, N_{x^-})$ is not greater than the original pessimistic error $U_{CF}(E, N)$, then it makes sense to delete the condition X. For each rule, the pessimistic error for removing each condition is calculated. If the lowest pessimistic error is not greater than that of the original rule, then the condition that gives the lowest pessimistic error is removed. The removal is repeated until the pessimistic error of the rule cannot be improved.

After this simplification, the set of rules can be exhaustive and redundant. For each class, only a subset of rules is chosen out of the set of rules classifying it. The subset is chosen based on the Minimum Description Length principle. The principle states that the best rule set should be the rule set that required the fewest bits to encode the rules and their exceptions. For each class, the encoding length for each possible subset of rules is estimated. The subset that gives the smallest encoding length is chosen as the rule set of that class.

2.3 Association Rule Mining

Association rule mining (Agrawal et al. [1993]) focuses on discovering knowledge between items in a large database of sales transactions. Association rule is a rule of the form "if X then Y", where X and Y are items in a transaction. Association

rule mining is different from classification, as there is no pre-specified classes in the consequent. An association rule is valid if it can satisfy the threshold requirement on confidence factor and support. The rule is required to have at least $c\%$ of records that satisfy X also satisfy Y, where c is the confidence threshold. It is also required that the number of records satisfying both X and Y has to be larger than $s\%$ of the records, where s is the support threshold.

The problem of mining association rules from a database can be solved in two steps. The first step is to find the sets of attributes that have enough support. These sets are called *large itemsets* as 'large' is used to denote having enough support. The second step is from each large itemset, association rules with confidence larger than the threshold are searched. The attributes are divided into antecedents and consequent and the confidence is calculated. The main researches (Agrawal et al. [1993]; Mannila et al. [1994]; Agrawal and Srikant [1994]; Han and Fu [1995]; Park et al. [1995]) consider Boolean association rules, where each attribute must be Boolean (e.g. have or have not buy the item). They focus on developing a fast algorithm for the first step, as this step is very time consuming. They can be efficiently applied to large databases, but the requirement of Boolean attributes limited their uses.

2.3.1 Apriori

Apriori (Agrawal and Srikant [1994]) is an algorithm for generating large itemsets (i.e. the first step) in Boolean association rule mining. The support of an itemset has a characteristic that the subsets of a large itemset must be large, and supersets of a small (i.e. not large) itemset cannot be large. Apriori makes use of this characteristic to drastically reduce the search space.

The outline of Apriori algorithm is listed as follows:

1. Count the support of item sets with 1 element.
2. L_1 = set of size 1 itemsets that are large.
3. for $(k = 2; k < no\_of\_attributes; k++)$

(a) generate extensions of each size $k-1$ large itemset by adding one more attributes;

(b) C_k = set of extensions of size $k-1$ large itemsets;

(c) For each itemset in C_k, if one of its size $k-1$ subset is not in L_k, delete it from C_k;

(d) For each itemset in C_k, count the support and check whether it is large;

(e) L_k = set of large itemsets in C_k.

Apriori first searches for large itemsets with one attribute. Then other large itemsets are searched from the itemsets known to be large. The large itemsets are extended by adding one attribute. If one subset of the extended itemset is not known to be large, this itemset is rejected because the subset of a large itemset must be large. The supports of these extended itemsets are counted to check whether they are still large. Once a large itemset is found to be not large, further extension of it is no longer necessary because its superset must be small.

2.3.2 Quantitative Association Rule Mining

Quantitative Association Rules do not restrict the attributes to be Boolean. Quantitative or categorical attributes are allowed. In Srikant and Agrawal [1996], the problem of mining quantitative association rules is mapped into a Boolean association rule problem. Intervals are made for each quantitative attribute. A new Boolean attribute is created for each interval or category. This attribute is set to 1 if the original attribute is in that interval or category. For example, a record with age equals 23 will have '1's in the new interval attributes 'Age:(21-25)' and 'Age:(15-30)', and have '0's in the new interval attribute 'Age:(15-20)', 'Age:(26-30)'. However, this mapping will face two new problems:

- "ExecTime". The number of attributes is hugely increased, and greatly affects the execution time.

- "ManyRules". If an interval of a quantitative attribute has minimum support, any range containing this interval will also has minimum support. Thus the number of rules increase greatly. Many of them just differ in the ranges of the quantitative attributes and in fact refer to the same association.

To tackle the first problem, a "maximum support" parameter is required from the user. The new Boolean attributes are not created for all possible intervals. If the support of an interval exceeds the maximum support, it will not be considered as the rule will be too general and should already be covered by other rules having a smaller interval. To tackle the second problem, an "interesting level" parameter is required from the user. An interesting measure is defined to measure how much the support and/or confidence of a rule is greater than expected. Those rules with interest measure lower than the user requirement is pruned.

2.4 Statistical Approach

Statistic and data mining both try to search knowledge from data. Statistic approach focuses more on quantitative analysis. A statistical perspective on knowledge discovery has been given in Elder IV and Pregibon [1996]. Statisticians usually assume a model for the data and then look for the best parameters for the model. They interpret the models based on the data. They may sacrifice some performance in order to be able to extract the meaning from the model. However in recent years statistician has also moved the objective to the selection of a suitable model. Moreover, statisticians place strong emphasis on estimating or explaining the model uncertainty by summarizing the randomness to a distribution. The uncertainties are captured in the standard error of the estimation.

2.4.1 Chi Square Test and Bayesian Classifier

One of the most useful statistical measures for data mining is the chi-square (χ^2) test described in equation 2.5. The value χ^2 measures the dependency between two

attributes. If this value is smaller than a certain threshold, it can conclude that one attribute is not relevant for determining the other attribute. The commonly used threshold is χ^2 at 95% or 99% confidence.

The Bayesian probability theorem can be used to classify an object into one of the classes $\{c_1, c_2, \cdots, c_m\}$. Let the object be described by a feature vector F which consists of attributes $\{f_1, f_2, \cdots, f_l\}$. The probability of this object belongs to class c_i is given by

$$p(c_i|F) = \frac{p(F|c_i)p(c_i)}{p(F)} \tag{2.10}$$

The use of this theorem can provide probabilistic knowledge for classifications of unseen objects. The object with a feature vector F can be classified to the class c_i which gives the maximum value on this probability. Since the denominator $p(F)$ appears in every probability, it is actually a normalizing factor and can be ignored in the calculation. The probability $p(c_i)$ can be estimated as the occurrence of c_i over the total number of existing objects. Thus the main concern is on how to estimate $p(F|c_i)$.

This probability can be estimated by making assumptions. The simplest assumption is that each feature in F is statistical independence, that is

$$p(F|c_i) = \prod_{k=1}^{l} p(f_k|c_i) \tag{2.11}$$

the value $p(f_k|c_i)$ can be estimated as the occurrence of objects in class c_i having f_k over the occurrence of objects in class c_i. Another assumption given in Wu et al. [1991] is that the probability can be under a normal distribution, that is

$$p(F|c_i) = \frac{1}{(2\pi)^{n/2}|C_i|^{1/2}} \exp(-\frac{1}{2}(F - M_i)'C_i^{-1}(F - M_i)) \tag{2.12}$$

where C_i is the covariance matrix and M_i is the mean vector over n unseen cases. Thus the problem is reduced to the measurement of the two parameters C_i and M_i.

	$A = a_1$	$A = a_2$	Total
$C = c_1$	o_{c_1,a_1}	o_{c_1,a_2}	o_{c_1}
$C = c_2$	o_{c_2,a_1}	o_{c_2,a_2}	o_{c_2}
Total	o_{a_1}	o_{a_2}	

Table 2.1: A contingency table for variable A vs. variable C

2.4.2 FORTY-NINER

FORTY-NINER (Zytkow and Baker [1991]) is a system for discovering regularities in a database. It searches for significant regularities compared to the null distribution hypothesis. The search is divided into two phases. The first phase is a search for two-dimensional regularities (i.e. regularities between two variables). The second phase generalizes the two-dimensional regularities to more dimensions. Either phase can be repeated many times with human interventions.

In the first phase, each attribute is transformed by using aggregation, slicing and projection. The search is performed on partitions of the database. The user can reduce the search space by limiting the number of independent variables and the depth of partitioning. The regularity is represented in a contingency table and in the best linear fit. An example of a contingency table is shown in Table 2.1, where o_{c_1,a_1} is the actual number of occurrence of $C = c_1$ and $A = a_1$. This value is compared with the expected occurence $e_{c_1,a_1} = o_{c_1} \times o_{a_1}/N$ (where N is the total number of records), and χ^2 is calculated to measure the significance of the regularity. The best liner fit between C and A is a linear regularity $C = mA + b$ obtained by using the least squares method, where m is the slope and b is the intercept. A value r^2 measures the significance of the linear regularity. It is calculated over all data points (X_i, Y_i) using the formula:

$$r^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2}} \tag{2.13}$$

where $\bar{Y}$ is the average value of Y over the n data points, and $\hat{Y}_i$ is the value of Y predicated by the linear regularity.

In the second phase, the user selects the 2-D regularities for expansions. The regularity expansion module adds one dimension at a time and the multi-dimension regularity is formed. This module can be applied recursively. Since the search space would be exponential if all possible multi-dimensional regularities is considered, user intervention is required to guide the search.

2.4.3 EXPLORA

EXPLORA (Hoschka and Klösgen [1991]; Klösgen [1993]) is an integrated system for helping the user to search for interesting relationships in the data. A statement is an interesting relationship between a value of a dependent variable and values of several independent variables. Various statement types are included in EXPLORA, e.g. rules, changes and trend analyses. The value of the dependent variable is called the *target group* and the combination of values of independent variables is called the *subgroup*. For example, the sufficient rule pattern

> 48% of the population are CLERICAL. However, 92% of AGE $>$ 40, SALARY $<$ 10260 are CLERICAL

is a relationship between the target group CLERICAL and the independent variables are AGE and SALARY. The user selects one statement type, identifies the target group and the independent variables, and inputs the suitable parameters. EXPLORA calculates the statistical significance of all possible statements and outputs the statements with significance above the threshold.

The search algorithm in EXPLORA is a graph search. Given a target group, EXPLORA search for the subgroup for regularities. It first uses values from one variable, then combinations of values from two variables, and then combination of values from three variables, and so on until the whole search space is exhaustively explored. The search space can be reduced by limiting the number of combinations of independent variables and by the use of redundancy filters. Depending on the type of the statements, different redundancy filters can be used. For example, for the sufficient rule pattern "If subgroup then target group", the redundancy filter

is "if a statement is true for a subgroup a, then all statements for the subgroup a $\wedge$ other values are not interesting". For the necessary rule pattern "If target group then subgroup", the redundancy filter is "if a statement is true for subgroup $a \wedge b$, then the statement for subgroup a is true".

2.5 Bayesian Network Learning

Bayesian network (Charniak [1991]; Heckerman and Wellman [1995]) is a formal knowledge representation supported by the well-developed Bayesian probability theory. A Bayesian network captures the conditional probabilities between attributes. It can be used to perform reasoning under uncertainty. A Bayesian network is a directed acyclic graph. Each node represents a domain variable, and each edge represents a dependency between two nodes. An edge from node A to node B can represent a causality, with A being the cause and B being the effect. The value of each variable should be discrete. Each node is associated with a set of parameters. Let N_i denote a node and Π_{N_i} denote the set of parents of N_i. The parameters of N_i are conditional probability distributions in the form of $P(N_i|\Pi_{N_i})$, with one distribution for each possible instance of Π_{N_i}. Figure 2.2 is an example Bayesian network given in Charniak [1991]. This network shows the relationships between whether the family is out of the house (fo), whether the outdoor light is turned on (lo), whether the dog has bowel problem (bp), whether the dog is in the backyard (do), and whether the dog barking is heard (hb).

Since a Bayesian network can represent the probabilistic relationships among variables, one possible approach of data mining is to learn a Bayesian network from the data (Heckerman [1996]; Heckerman [1997]). The main task of learning a Bayesian network is to automatically find directed edges between the nodes, such that the network can best describe the causalities. Once the network structure is constructed, the conditional probabilities are calculated based on the data. The problem of Bayesian network learning is computationally intractable (Cooper [1990]). However, Bayesian networks learning can be implemented by imposing

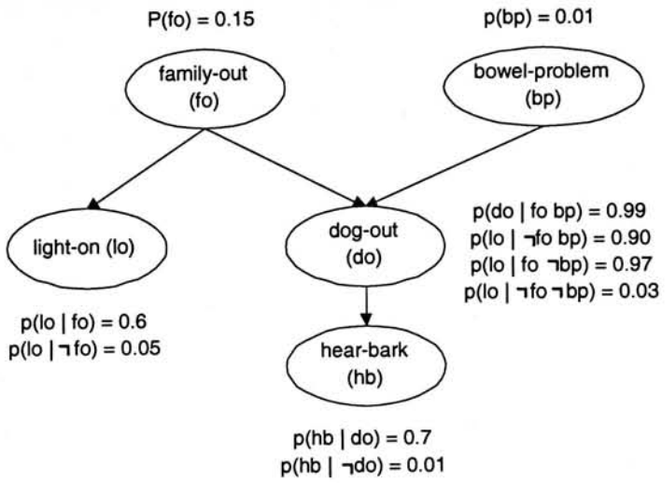


Figure 2.2: A Bayesian network example

limitations and assumptions. For instance, the algorithms of Chow and Liu [1968] and Rebane and Pearl [1989] can learn networks with tree structures, while the algorithms of Herskovits and Cooper [1990], Cooper and Herskovits [1992] and Bouckaert [1994] require the variables to have a total ordering. More general algorithms include Heckerman et al. [1995], Spirtes et al. [1993] and Singh and Valtorta [1993]. More recently, Larranaga et al. [1996a]; Larranaga et al. [1996b] has proposed algorithms for learning Bayesian networks using Genetic Algorithm.

2.5.1 Learning Bayesian Networks using the Minimum Description Length (MDL) Principle

One approach for Bayesian network learning is to apply the *Minimum Description Length* (MDL) principle (Lam and Bacchus [1994]; Lam [1998]). In general there is a trade-off between accuracy and usefulness in the construction of a Bayesian network. A more complex network is more accurate, but computationally and conceptually more difficult to use. Nevertheless, a complex network is only accurate for the training data, but may not be able to uncover the true probability distribution. Thus it is reasonable to prefer a model that is more useful. The MDL principle (Rissanen [1978]) is applied to make this trade-off. This principle

states that the best model of a collection of data is the one that minimizes the sum of the encoding lengths of the data and the model itself. The MDL metric measures the *total description length* DL of a network structure G. A better network has a smaller value on this metric. A heuristic search can be performed to search for a network that has a low value on this metric.

Let $U = \{X_1, \ldots, X_n\}$ denote the set of nodes in the network (and thus the set of variables, since each node represents a variable), Π_{X_i} denote the set of parents of node X_i, and D denote the training data. The total description length of a network is the sum of description lengths of each node:

$$DL(U, G, D) = \sum_{X_i \in U} DL(X_i, \Pi_{X_i}) \tag{2.14}$$

This length is based on two components, the *network description length* DL_{net} and the *data description length* DL_{data}:

$$DL(X_i, \Pi_{X_i}) = DL_{net}(X_i, \Pi_{X_i}) + DL_{data}(X_i, \Pi_{X_i}) \tag{2.15}$$

The formula for the network description length is

$$DL_{net}(X_i,\Pi_{X_i}) = k_i \log_2(n) + d(s_i - 1) \prod_{j \in \Pi_{X_i}} s_j \tag{2.16}$$

where k_i is the number of parents of variable X_i, s_i is the number of values X_i can take on, s_j is the number of values a particular variable in Π_{X_i} can take on, and d is the number of bits required to store a numerical value. This is the description length for encoding the network structure. The first part is the length for encoding the parents, while the second part is the length for encoding the probability parameters. This length can measure the simplicity of the network.

The formula for the data description length is

$$DL_{data}(X_i,\Pi_{X_i}) = \sum_{X_i,\Pi_{X_i}} M(X_i,\Pi_{X_i}) \log_2 \frac{M(\Pi_{X_i})}{M(X_i,\Pi_{X_i})} \tag{2.17}$$

where $M(.)$ is the number of cases that match a particular instantiation in the database. This is the description length for encoding the data. A Huffman code is used to encode the data using the probability measures defined by the network. This length can measure the accuracy of the network.

2.5.2 Discretizating Continuous Attributes while Learning Bayesian Networks

Bayesian network can only represent discrete variables. One approach to handle the databases with continuous variables is to discretize them first. The continuous variables are usually discretized by thresholds specified by human. However, different discretization policy will produce different network structure. The causality will be lost if the discretization is not suitable. Thus it is desirable to search for the best discretization policy before the learning of the Bayesian network is performed.

Formally, a *discretization sequence* λ defines a function that maps a continuous variable to a discrete variable. Each discretization sequence contains a list of threshold values. The variable will be discretized according to the ranges specified by the thresholds. For example, if the threshold list is $< t_1, t_2, \cdots, t_k >$, $t_1 < t_2 \cdots < t_k$, the function f_λ defined in the discretization sequence λ should be:

$$f_\lambda(x) = \begin{cases} 0 & \text{if } x < t_1 \\ i & \text{if } t_i \leq x < t_{i+1} \\ k & \text{if } t_k \leq x \end{cases}$$

A *discretization policy* , $\Lambda = \{\lambda_i : X_i \text{ is continuous}\}$, is a collection of discretization sequences for each continuous variable. The policy defines a new set of variables $U^* = \{X_1^*, \ldots, X_n^*\}$ where $X_i^* = f_{\lambda_i}(X_i)$ if X_i is continuous and $X_i^* = X_i$ otherwise.

Friedman and Goldszmidt [1996] extended the MDL score to evaluate the

discretization policy while learning the Bayesian network structure. The original training data D is discretized into a new data set D^*. A Bayesian network structure G for the discretized variables U^* can be learned from D^*. The new definition of the MDL score includes the description length of the network as well as description length of the discretization policy:

$$DL^*(U^*, G, \Lambda, D) = DL(U^*, G, D^*) + DL_\Lambda(\Lambda) + DL_{D^* \to D}(D, \Lambda) \qquad (2.18)$$

- The first part, $DL(U^*, G, D^*)$, is the score of the network under the discretized data, and can be calculated by using Equation 2.14.

- The second part, $DL_\Lambda(\Lambda)$, is the length for encoding the particular discretization policy Λ over all of the possible discretization policy. Let $Val_D(X_i)$ to be the set of values of X_i that appear in the data set D, $s_i = |Val_D(X_i)|$ to be the cardinality of this set, and $s_i^* = |Val_D(X^*)|$ to be the cardinality of the set of values of X_i^*. The thresholds for X_i in the discretization policy is chosen from among the $s_i - 1$ mid-point values. Since there are $\binom{s_i-1}{s_i^*-1}$ different discretization sequences of cardinality s_i^*, the discretization sequence can be indexed by using $\log \binom{s_i-1}{s_i^*-1}$ bits. Because $\log \binom{s_i-1}{s_i^*-1} \leq (s_i - 1)H(\frac{s_i^*-1}{s_i-1})$, where $H(p) = -p \log p - (1-p)\log(1-p)$, the description length of Λ is equal to:

$$DL_\Lambda(\Lambda) = \sum_{X_i \text{ is continuous}} (s_i - 1)H(\frac{s_i^* - 1}{s_i - 1}) \qquad (2.19)$$

- The third part, $DL_{D^* \to D}(D, \Lambda)$, is the encoding length for reconstruct U from U^*. For a particular value of X_i, the encoding length for reconstruction from X_i^* using the Huffman code is approximately $-\log p(X_i|X_i^*) = -\log(\frac{M(X_i)}{M(X_i^*)})$, where $M(.)$ is the number of cases that match a particular instantiation in the database. This encoding has to be repeated for each

record in the database. Thus this part is equal to:

$$DL_{D^* \to D}(D, \Lambda) = -\sum_i \sum_{X_i} M(X_i) \log(\frac{M(X_i)}{M(X_i^*)}) \tag{2.20}$$

Friedman and Goldszmidt [1996] have also described a greedy approach for learning the discretization policy as well as the Bayesian network. The approach learns the discretization policy and the network structure alternatively. It starts with a initial discretization policy and learns the Bayesian network from the discretized data set by using the MDL metric. Based on this learned structure, a discretization policy is learned by using the MDL metric. In learning the discretization policy, only one variable is re-discretized at a time, with the discretization for other variables being fixed. The discretization sequence of this variable is reset to empty (i.e. no threshold values) first. The greedy approach searches for a possible refinement. The split that gives the largest decrease in the MDL metric is added to the current discretization sequence. The process is repeated until there is no improvement. The algorithm of this approach can be summarized below:

1. Start with an initial discretization policy.
2. Learn a network structure from the discretized data set.
3. Learn a new discretization policy based on the learned network structure.
   3.1 For each variable, search for the best discretization sequence.
      3.1.1 Reset the discretization sequence to empty.
      3.1.2 Calculate the decrease in MDL for each possible split
      3.1.3 Add the split with the largest decrease to the current discretization sequence.
      3.1.4 Repeat 3.1.2-3.1.3 until no improvement.
   3.2 Repeat 3.1 until no improvement in MDL.
4. Repeat 2-3 until no improvement in MDL.

Chapter 3

Overview of Evolutionary Computation

3.1 Evolutionary Computation

Evolutionary Computation is a term to describe computational methods that simulate the natural evolution to perform function optimization and machine learning. A potential solution to the problem is encoded as an *individual.* An evolutionary algorithm maintains a group of individuals, called the *population*, to explore the search space. A *fitness function* evaluates the performance of each individual to measure how close it is to the solution. The search space is explored by evolving new individuals. The evolution is based on the Darwinian principle of evolution through natural selection: the fitter individual has a higher chance of survival, and tends to pass on its favorable traits to its offspring. A 'good' parent is assumed to be able to produce 'good' or even better offspring. Thus an individual with a higher score in the fitness function have a higher chance of undergoing evolution. Evolution is performed by changing the existing individuals. New individuals are generated by applying *genetic operators* that alter the underlying structure of individuals.

This search technique is a 'weak' method. It is a general, domain independent method that does not require any domain-specific heuristic to guide the search.

Parameter values: $x_1 = 7, x_2 = 5, x_3 = 1$
Binary values: $x_1 = 111, x_2 = 101, x_3 = 01$
Chromosome:

1	1	1	1	0	1	0	1

Figure 3.1: The chromosome in GA

Examples of algorithms in evolutionary computation include Genetic Algorithm, Genetic Programming, Evolutionary Programming and Evolution Strategy. They mainly differ in the evolution models assumed, the evolutionary operators employed, the selection methods, and the fitness functions used.

3.1.1 Genetic Algorithm

Genetic Algorithms (GA) (Holland [1992]; Goldberg [1989]) is a search method for optimization. The goal of GA is to search for values for parameters $x_1, x_2, \ldots, x_n$ that optimizes a fitness function, $f(x_1, x_2, \ldots, x_n)$. The values of parameters are encoded as a fixed-length binary bit string, which becomes the *chromosome* of an individual. For example, if the parameters are real numbers, the binary value of these parameters can be concatenated to form the chromosome, as illustrated in 3.1. Each individual stores one chromosome. The binary bit string is called the *genotype* of the individual, while the parameter values encoded by the bit string is called the *phenotype* of the individual.

The algorithm of a simple GA is shown in Table 3.1. The algorithm begins with an initial population of individuals. The chromosomes of these individuals are randomly generated. Each individual is then evaluated by a fitness function to get a fitness value. The binary bits in the chromosome are decoded and the value of fitness function on this set of parameter values is calculated. Then a number of generations are iterated to evolve better individuals. In each generation, certain individuals are selected from the population of current generation as the parents. The selection is based on the Darwin's principle of survival of the fittest. The probability of an individual being selected is proportional to the fitness of the individual. This selection method is called fitness proportionate selection. The detail of selection methods is discussed in Section 3.1.5. Crossover is performed

```
Initialize the generation, t, to be 0.
Initialize a population of individual, Pop(t), with size popsize
Evaluate the fitness of all individual in Pop(t)
While the termination criteria is not satisfied
    Initialize Pop(t+1) as an empty set
    While size of Pop(t+1) < popsize
        Select two individuals, parent1 and parent2, from Pop(t)
        Cross-over parent1 and parent2 to produce child1 and child2
        Mutate child1 and child2
        Evaluate the fitness of child1 and child2
        Put child1 and child2 into Pop(t+1)
    Increase the generation t by 1
Return the individual with the highest fitness value
```

Table 3.1: The Simple Genetic Algorithm

with a probability of p_c to recombine two parents. If crossover is not performed, then the children is just the same as the parents. The children then further undergo a mutation with a probability of p_m. The mutated children are put into the next generation of population. The generation is iterated until the termination criterion is met. An example of a termination criterion is that an individual can achieved a requirement of fitness value, or the maximum number of generation is exceeded.

Crossover exchanges the genetic materials in the chromosomes of two parents to produce two children. A random position in the bit string is chosen. The bits after this crossover point in the parental chromosomes are exchanged, as illustrated in Figure 3.2. This kind of crossover is called one point crossover. Mutation flips a bit from 0 to 1 or vice versa, as illustrated in Figure 3.3. Each bit has the same probability p_m of mutation. Mutation is a secondary operator that can restore lost genetic materials. For example, if all the individuals with 0 in the first bit are not selected as parents, then only crossover cannot re-generate a 0 at the first bit. However, mutation can re-introduce this lost 'gene' into the population.

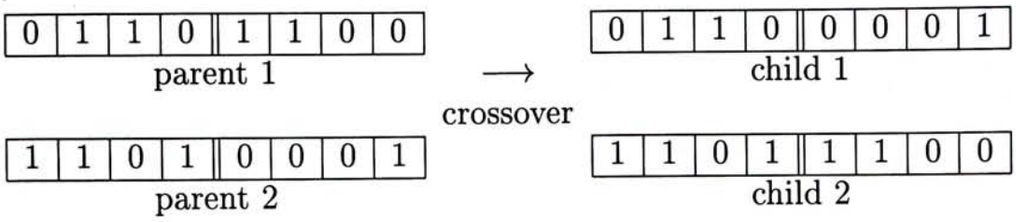


Figure 3.2: Crossover in GA. The crossover point is the 4th bit and the bits after it are exchanged

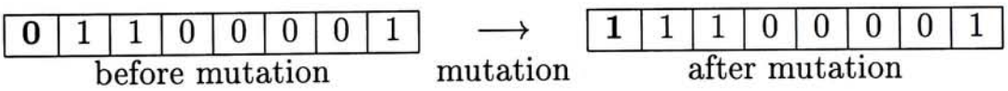


Figure 3.3: Mutation in GA. Mutation occurs at the 1st bit and the 4th bit

3.1.2 Genetic Programming

Genetic Programming (GP) (Koza [1992]; Koza [1994]) is an extension of Genetic Algorithm. They mainly differ on the representation of chromosomes. The chromosome of GA is with fixed length. Each bit in the chromosome has its own meaning. The chromosome of GP is a tree consists of functions and terminals. The phenotype of the chromosome is a computer program, which when executed can solve the problem.

GP evolves a computer program in the language LISP. In LISP, all operations are executed by performing functions to arguments. A function call is represented as a list of the function and the arguments, enclosed by parentheses. The first element in the list is the function and the subsequent elements are the arguments. This kind of expression is called a S-expression. Every S-expression can be represented in a tree format. A function becomes a parent node and the arguments become the branches. For example, Figure 3.4 is the tree representing the S-expression `(+ 1 2 (IF (> TIME 10) 3 4)`. The function `IF` returns the second argument if the first argument is true, otherwise the third argument. The symbol `TIME` is a variable. The internal nodes of this tree are the functions and the leaf nodes are the terminals. This tree representation is the knowledge representation of chromosomes used in GP.

To apply GP to a problem, a set of functions F and a set of terminals T have to be defined. The algorithm of GP is very similar to GA. A set of initial individuals

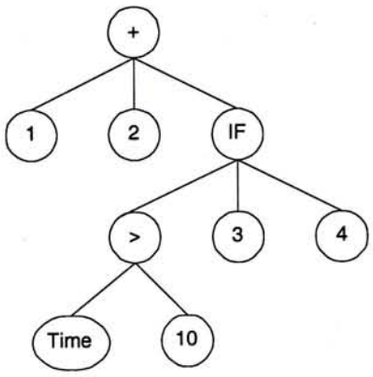


Figure 3.4: The tree representation of a S-expression

```
Initialize the generation, t, to be 0.
Initialize a population of individual, Pop(t), with size popsize
While the termination criteria is not satisfied
    Evaluate the fitness of all individuals in Pop(t)
    Initialize Pop(t+1) as an empty set
    While size of Pop(t+1) < popsize
        Choose a genetic operation probabilistically
        If reproduction
            Select one individual based on fitness
            Copy the individual into Pop(t+1)
        If crossover
            Select two individuals based on fitness
            Perform crossover
            Insert the two offspring into Pop(t+1)
        If mutation
            Select one individual based on fitness
            Perform mutation
            Insert the offspring into Pop(t+1)
    Increase the generation t by 1
Return the individual with the highest fitness value
```

Table 3.2: The Algorithm of Genetic Programming

are created randomly from the function set and the terminal set. Each individual is evaluated by a fitness function. New individuals are evolved by genetic operators, including reproduction, crossover and mutation. The generation of evolutions repeated until the termination criterion is satisfied. The algorithm is sketched in Table 3.2.

To create an individual, a function is selected from F to be the root. A number of branches, which equals to the arity of this function, are created from the root. At each branch a symbol is selected from the set $F \cup T$. If a function is selected, the above process repeated recursively.

The genetic operators typically used in GP are reproduction, crossover and mutation. In reproduction, the parent is just copied unchanged to the new population. In crossover, two subtrees are selected from the trees of each parent. These subtrees are exchanged to produce two children, as shown in Figure 3.5. In mutation, a subtree is selected from the parental tree, and then replaced by a randomly generated subtree, as shown in Figure 3.6. The generation of the replacing subtree is the same as the generation of the initial population. Mutation is considered as less important in GP. It is because particular functions and terminals are not associated with fixed positions. It is rare for a function or terminal to disappear entirely from all the nodes of all individuals. Thus, mutation is not a necessary operation to restore the lost genetic materials.

3.1.3 Evolutionary Programming

Evolutionary Programming (EP) (Fogel [1994]; Fogel et al. [1966]) emphasizes on the behavioral linkage between parents and their offspring, rather than seeking to emulate specific genetic operators as observed in nature. Different from GA, EP does not require any specific genotype in the individual. EP employs a model of evolution at a higher abstraction. Mutation is the only operator used for evolution.

A typical process of EP is outlined in Table 3.3. A set of individuals is randomly created to make up the initial population. Each individual is evaluated by the fitness function. Then each individual produces a child by mutation. There

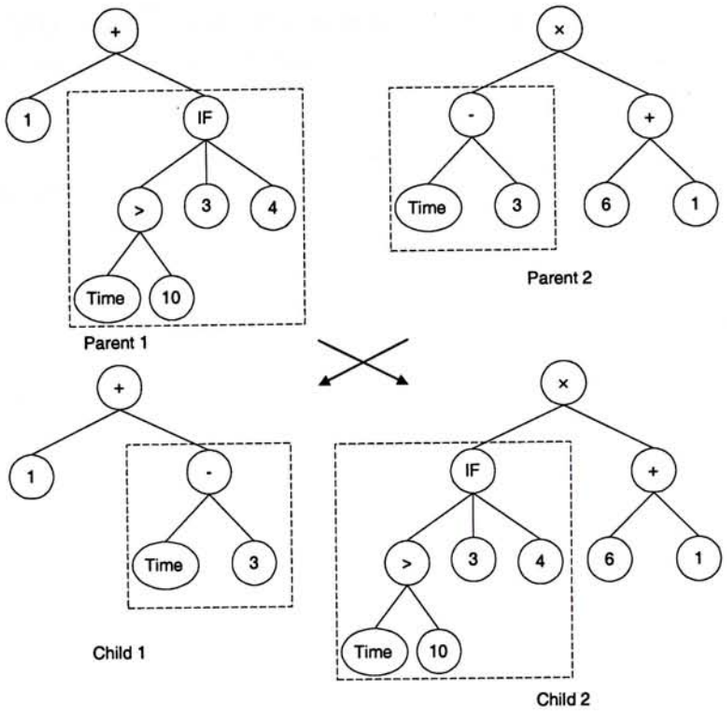


Figure 3.5: An example of crossover in GP. The selected subtree is enclosed by the dashed box

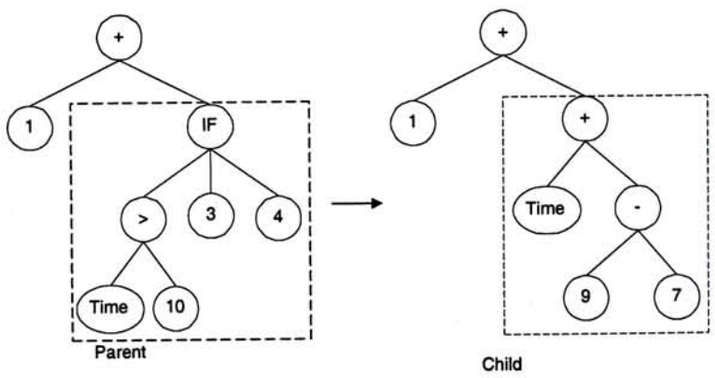


Figure 3.6: An example of mutation in GP. The selected subtree is enclosed by the dashed box

```
Initialize the generation, t, to be 0.
Initialize a population of individual, Pop(t)
Evaluate the fitness of all individual in Pop(t)
While the termination criteria is not satisfied
    Produce one or more offspring from each individual by mutation
    Evaluate the fitness of each offspring
    Perform a tournament for each individual
    Put the individuals with high tournament scores into Pop(t+1)
    Increase the generation t by 1
Return the individual with the highest fitness value
```

Table 3.3: The Algorithm of Evolutionary Programming

is a distribution of different types of mutation, ranging from minor to extreme. Minor modifications in the behavior of the offspring occur more frequently and substantial modifications occur more unlikely. The offspring is also evaluated by fitness function. Then tournaments are performed to select the individuals for the next generation. For each individual, a number of rivals are selected among the parents and offspring. The tournament score of the individual is the number of rivals with lower fitness scores than itself. Individuals with higher tournament scores are selected as the population of next generation. There is no requirement that the population size is held constant. The process is iterated until the termination criterion is satisfied.

EP has two characteristics. First, there is no constraint on the representation. Mutation operator does not demand a particular genotype. The representation can follow from the problem. Second, mutations in EP attempt to preserve behavioral similarity between offspring and their parents. An offspring is generally similar to its parent at the behavioral level with slight variations. EP assumes that the distribution of potential offspring is under a normal distribution around the parent's behavior. Thus, the severity of mutations is according to a statistical distribution.

3.1.4 Evolution Strategy

Evolution Strategy (ES) (Rechenberg [1973]; Schwefel [1981]) is originally designed for real-valued function optimization. It emphasizes on the individual, i.e. the phenotype, to be the object to be optimized. Each parameter is represented as an object variable x_j. Each x_j is associated with a strategy variable σ_j, which controls the degree of mutation to x_j. The genotype of an individual is a vector of pairs (x_j, σ_j).

There are various models of evolution strategy. In $(\mu + \lambda)$-ES, the population size is μ, and λ more individuals are evolved in each generation by recombination and mutation. Among these $(\mu + \lambda)$ individuals, only the best μ individuals are kept in the population. The selection is based on the score of an objective function F. The evolution terminates when the optimal set of values for all the objective variables are found, or when the maximum number of generations is reached.

There are various methods of recombination, and can be classified as non-global and global. In non-global combination, two individuals are selected as parents. For non-global discrete recombination, the value of each pair (x_j, σ_j) of the offspring is selected randomly from one of the parents. For non-global intermediate recombination, the value of each pair (x_j, σ_j) of the offspring is set to the mean value of the two parents. On the other hand, in global recombination, a pair of parents are selected for *each* pair of (x_j, σ_j). Thus if the individual contains L pairs of (x_j, σ_j), L pairs of parents are selected. For global discrete recombination, the value of each pair (x_j, σ_j) of the offspring is selected randomly from one of its parents. For global intermediate recombination, the value of each pair (x_j, σ_j) of the offspring is set to the mean value of its parents.

Mutation modifies the value of each x_j as well as each σ_j. According to the biological observation, offspring are similar to their parents and that smaller modifications occur more frequently than larger modifications. Thus the new value of x_j after mutation, x'_j, is equal to:

$$x'_j = x_j + N(0, \sigma_j)$$

where $N(0, \sigma_j)$ is a Gaussian random number with mean 0 and standard derivation σ_j. A mutation is regarded as successful if the mutated individual has a higher score on F than the parent. The ratio r is the ratio of successful mutations to all mutations. It is observed that the convergence rate is optimal if r equals to $1/5$. Thus the new value of σ_j of each individual, σ'_j, is changed based on r:

$$\sigma'_j = \begin{cases} c_d \sigma_j & \text{if } r < 1/5 \\ c_i \sigma_j & \text{if } r > 1/5 \\ \sigma_j & \text{if } r = 1/5 \end{cases}$$

where c_d and c_i are constants. If r is smaller than $1/5$, σ is decreased by multiplying a constant $c_d < 1$, so as to generate offspring closer to the parents. If r is larger than $1/5$, σ is increased by multiplying a constant $c_i > 1$, so as to broaden the search.

ES and EP both use a statistical distribution of mutations. However, ES typically uses deterministic selection that the worst individuals are eliminated, while EP typically uses a stochastic tournament selection. EP is an abstraction of evolution at the level of *species*. Thus no recombination is used because recombination does not occur between species. In contrast, ES is an abstraction of evolution at the level of individual behavior and hence recombination is reasonable.

3.1.5 Selection Methods

The classical method for selection of parents is the fitness proportionate selection (Holland [1992]), or called the 'roulette wheel' selection. The individuals in the population form a roulette wheel, where each individual has a slot sized in proportion to its fitness. The roulette wheel is turned to select the parent. Thus the probability of the ith individual being selected is $f_i / \sum_i f_i$, where f_i is the fitness of the ith individual. However, there is a deficiency in this selection method. In the early generations, a few individuals may have extraordinarily high fitness values. Fitness proportionate selection allocates a large number of offspring to these

individuals, and cause premature convergence. At the later stages, the individuals may have very close fitness values. Fitness proportionate selection cannot differentiate the better individuals and allocates an almost equal number of offspring to all individuals.

Alternative selection methods have been proposed. In the rank selection method (Baker [1985]), the population is sorted according to the fitness. The probability of an individual being selected is inversely proportional to its rank, with the better one getting a higher chance. For example, the probability for selecting an individual can be $(N+1-r_i)/\sum_{k=1}^{N} k$, where N is the population size and r_i is the rank of the individual. This selection method gives less emphasis on comparatively high-fitness individuals. On the other hands, it can distinguish individuals with a slightly difference in the fitness scores. In the tournament selection method, a group of individuals with size q are selected from the population. Among this group, the individual with the highest fitness value is selected. This selection method simulates the phenomenon that several individuals fight over the right of mating. However in these two methods, the probability of selection is not directly linked with the value of the objective function for optimization.

3.2 Generic Genetic Programming

Pure GP does not make any distinction between all the functions and terminals. It requires the function set and terminal set to have the closure property: All the functions in the function set should be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set (Koza [1992]). Some operations must be modified before being used in GP. For example, division must be modified so that its value is defined when the denominator is zero. Another example is the commonly used operator '=', which tests the equality of two numbers. It does not fulfill the closure property as its return value should be with Boolean type but the arguments it takes are

Expr → (if *Boolean Real Real*)
Boolean → (*Operator Real Real*)
Boolean → T | F
Operator → = | < | > | <= | >=
Real → var1 | var2 | var3
Real → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Table 3.4: An example grammar. The symbol `if` returns the second argument if the first argument is true, or else the third argument

real numbers. One solution is to modify the operator such that it returns a real number 1 for the value 'true' and returns 0 for the value 'false'. But this brings out other problems. For example, if the operators {+, -, AND, =} are used together in the function set, it may produce meaningless programs like "$(x$ AND $y) + (x = y)$". The closure requirement greatly limits the representation power of genetic programming.

Generic Genetic Programming (Wong and Leung [1995]; Wong and Leung [1997]; Wong [1995]) (GGP) extends GP further to increase the consistency and flexibility. GGP uses a grammar to control the placement of functions and terminals. A function or a terminal must be placed in a position that conforms to the grammar. The genotype used in GGP is a derivation tree instead of the tree representation of S-expression in GP.

A grammar G is a 4-tuple $G = (V_N, V_T, P, X)$ where V_N is a finite set of *non-terminal* symbols, V_T is a finite set of *terminal* symbols, P is a set of *production rules* of the form $\alpha \rightarrow \beta$, and $X \in V_N$ is the *start symbol* of G. A production rule in the form $\alpha \rightarrow \beta \mid \gamma$ denotes two grammar rules $\{\alpha \rightarrow \beta,\ \alpha \rightarrow \gamma\}$. Table 3.4 is an example grammar. The start symbol is `Expr`, the italic terms are non-terminals and other terms are terminals.

If there is a production rule $\alpha \rightarrow \beta$, then the symbol α can be rewritten as β. This *rewrite* is denoted by $\alpha \Rightarrow \beta$. A *derivation* is zero or more rewrite steps. A *complete derivation* is a derivation from the start symbol such that there are only non-terminals in it. Table 3.5 shows an example of a complete derivation of the grammar listed in Table 3.4. The derivation process can be represented in a

Expr
⇒ (if *Boolean Real Real*)
⇒ (if (*Operation Real Real*) *Real Real*)
⇒ (if (> *Real Real*) *Real Real*)
⇒ (if (> var1 *Real*) *Real Real*)
⇒ (if (> var1 9) *Real Real*)
⇒ (if (> var1 9) 3 *Real*)
⇒ (if (> var1 9) 3 4)

Table 3.5: An example derivation

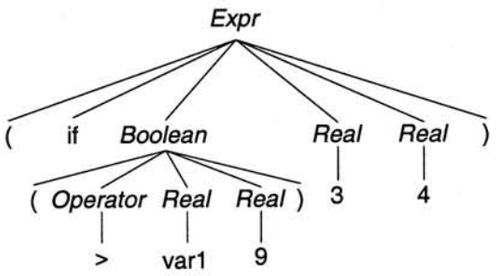


Figure 3.7: A derivation tree stored inside an individual of GGP

derivation tree. One rewrite step corresponds to one branching in the tree. Figure 3.7 is the derivation tree for the derivation in Table 3.5.

The grammar is used to generate individuals in Generic Genetic Programming. Each individual stores a derivation tree as in Figure 3.7. An individual is initially created by performing a complete derivation using the given grammar. Choices are randomly made if there are more than one possible derivation.

Similar to GP, there are three genetic operators in GGP. Reproduction copies one individual into the new population. Crossover differs from GP that it produces just one offspring from two parents. One parent is designated as the primary parent and the other is designated as the secondary parent. A subtree of the primary parental derivation tree is selected for crossover. It is then replaced by a subtree selected from the secondary parent. Figure 3.8 is an example of crossover. But the choice of the replacing subtree is restricted so that the grammar cannot be violated. A validation check is made to ensure that replacing the subtree can still obey the grammar. If the replacement is not valid, another subtree from the secondary parent will be selected. For example, Figure 3.9 shows part of a

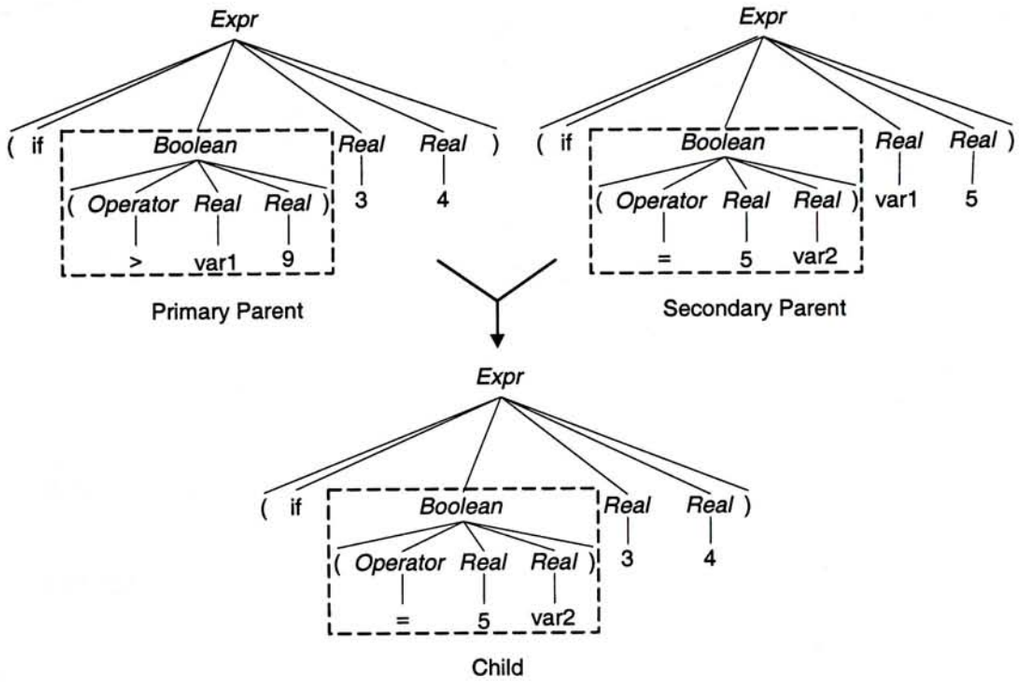


Figure 3.8: Crossover in Generic Genetic Programming

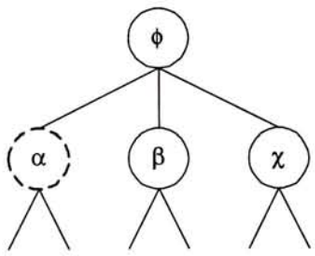


Figure 3.9: Part of a derivation tree

derivation tree. This subtree is valid if there is a grammar rule $\{\phi \rightarrow \alpha\beta\chi\}$. If the subtree at the node α is selected for crossover and replaced by a subtree starting with the node γ, the replacement is valid only if there exists a grammar rule $\{\phi \rightarrow \gamma\beta\chi\}$.

Mutation replaces a subtree in the derivation tree by a randomly generated subtree. A node in the derivation tree of the parent is selected. Each node corresponds to a symbol, and the grammar is used to derive another subtree rooting with this symbol. This new tree is used to replace the subtree at the selected node. Again, a check is needed to make sure the new tree evolved does not violate the given grammar. Figure 3.10 is an example of mutation in GGP.

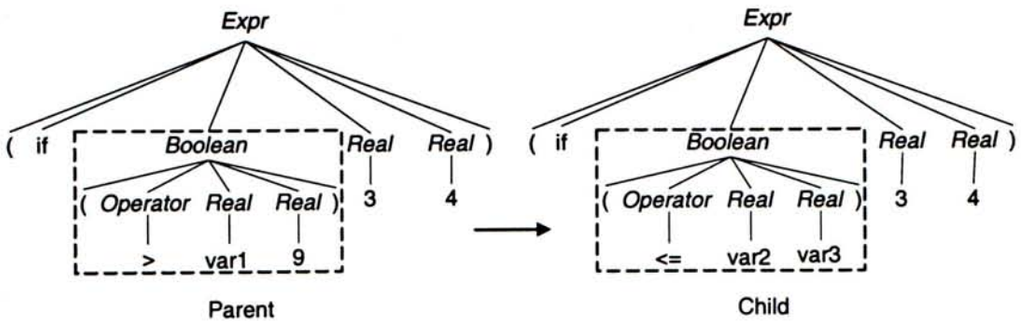


Figure 3.10: Mutation in Generic Genetic Programming

3.3 Data mining using Evolutionary Computation

Data mining can be considered as an optimization problem, which tries to search for the most accurate information from all possible hypotheses. Several systems have been built for learning concepts using evolutionary computation. GA can be used as the search algorithm by encoding a description of a concept into a bit string. However, the fixed-length chromosome in GA limited the representation of concept.

GABIL (De Jong et al. [1993]) uses a flat string representation to encode classification rules in disjunctive normal form (DNF). It uses the Pittsburgh's approach (Smith [1980]; Smith [1983]) that a single individual contains all the necessary descriptions for a concept and corresponds to a set of rules. Each individual is a variable-length string representing a set of rules. Each rule has a fixed length and consists of one test for each feature. The system uses k bits for the k values of a nominal feature. For example, the bit string in Table 3.6 represents the rule "if (F1 = 1 or 2 or 3) and (F2 = 1) then (class = 0)". Adaptive GABIL can adaptively allow or prohibit certain genetic operations for certain individuals. Extra bits are introduced to control the uses of certain genetic operations. These bits are also parts of evolution in GA.

GIL (Janikow [1993]) also used the Pittsburgh's approach. The bit string of

F1			F2				Class
1	1	1	1	0	0	0	0

Table 3.6: Bit string in GABIL

an individual represents a rule in multiple-valued logic language VL_1. It utilizes 14 genetic operators, such as rules exchange, new event, rules drops, rule split, condition drop, condition introduce, reference change and etc. These operators perform generalization, specialization or other modifications to the individuals in the rule set level, the rule level and the condition level.

In REGAL (Giordana and Neri [1995]), each individual encodes a disjunct consists of a conjunctive formula. Each individual is only a partial solutions, and the whole population is a redundant set of these partial solutions. An individual encodes a concept represented in the first-order logic, which is a language with variables. Several good individauls co-exist in the population by the use of a selection operator called Universal Suffrage operator to select the parents. At each generation, a set of examples is selected. The individuals covering a selected example are collected into a set. This set corresponds to a roulette wheel and a spin is made to select a winning individual. The winning individual from the selected examples becomes the parents. A parallel model is designed to enhance the execution speed.

GP can perform data mining by learning a program for classification. An example is the approach developed by Tackett [1993]. It uses a function set of $(+, -, \times, \div)$ and the conditional operator $\leq$, and a terminal set of all the 20 input features plus a random floating point constant. A program is evolved by GP. If the program returns a value larger than or equal to 0 given an input case, the input case is classified as a target. Otherwise it is classified as a non-target. Since the learned program is human understandable, knowledge can be obtained by examining the program. However, the program can be very complicated and difficult to interpret.

Chapter 4

Applying Generic Genetic Programming for Rule Learning

Rules are statements in the format of "if *antecedents* then *consequent*". Rules are commonly used by human to represent knowledge. Rule learning tries to learn rules from a set of data. It can be modeled as a search problem to search for the best rules. The search space can be very large depending on the rule representation. A powerful search algorithm is required. Generic Genetic Programming can be used as a possible approach. This chapter introduces how the problem of rule learning is modeled such that GGP can be applied.

To apply GGP, firstly a suitable representation has to be made to encode a rule as an individual. In GGP, a derivation tree is used to represent an individual, so a grammar for rules has to be design to create the derivation tree. Secondly, a set of suitable genetic operators has to be designed to evolve new individuals. Thirdly we have to design a suitable evaluation function to evaluate how good an individual is. This chapter introduces these three issues. The detail techniques for learning a set of rules are discussed in Chapter 5.

4.1 Grammar

The grammar of GGP governs the structures to be evolved. Rule learning can be achieved in GGP by using a suitable grammar that make up a rule. The grammar should specify the structure of a rule. The grammar specifies that a rule is of the form "if *antecedents* then *consequent*". The format of rules in each problem can be different. Thus for each problem, a specific grammar is written so that the format of the rules can best fit the domain. However, in general, the antecedent part is a conjunction of attribute descriptors. The consequent part is an attribute descriptor as well. An attribute descriptor characterizes an attribute. An attribute can be described in many ways, thus there are many different formats of descriptors. A descriptor can assign a value to a nominal attribute, a range of values to a continuous attribute, or can be used to compare attribute values.

GGP provides a powerful knowledge representation and allows a great flexibility on the rule format. The representation of rules is not fixed but depends on the grammar. Most of the rule learning methods can only learn a particular format of rules, for examples, rules with descriptors that compare the attributes with values. However, GGP allows a large variation in the attribute description. Rules with different formats can be learned, provided that the suitable grammar is supplied. Moreover, rules with the user desired structure can be learned because the user can specify the required rule format in the grammar.

An example is used to illustrate the use of grammar to represent the suitable rule format. Consider a database with 4 attributes. We want to learn rules about `attr4`, which is Boolean. The attribute `attr1` is nominal and coded with 0, 1 or 2. The attribute `attr2` is continuous between 0-200 and can be categorized into high, medium or low. The domain of `attr3` is identical to `attr2` and thus it is possible for the rule to compare them.

An example of the context free grammar for this database in given in Table 4.1. The symbols `erc1`, `erc2`, `erc3`, `boolean_erc` and `category_erc` in this grammar

Rule → if *Antes* , then *Consq* .
Antes → *Attr1* and *Attr2* and *Attr3*
Attr1 → any | *Attr1_descriptor*
Attr2 → any | *Attr2_descriptor*
Attr3 → any | *Attr3_descriptor*
Attr1_descriptor → attr1 = erc1
Attr2_descriptor → attr2 is category_erc
Attr2_descriptor → attr2 between erc2 erc2
Attr3_descriptor → attr3 *Comparator Attr3_term*
Comparator → = | ≠ | <= | >= | < | >
Attr3_term → attr2 | erc3
Consq → *Attr4_descriptor*
Attr4_descriptor → attr4 = boolean_erc

Table 4.1: An example grammar for rule learning.

are ephemeral random constants (ERCs). Each ERC has it own range for instantiation: `erc1` is within {0,1,2}, `erc2` and `erc3` is between 0-200, `boolean_erc` can only be T or F, `category_erc` can be either high, medium or low. The symbol 'any' serves as a 'don't care' in the rule. An attribute will not be considered in the rule if its attribute descriptor is 'any'. In this grammar, each attribute can be described by a descriptor in the rule, or by 'any' such that it is ignored by the rule. The attribute `attr1` have only one form of descriptor. The attribute `attr2` can have two forms of descriptors: it can be described by a range or by the category it belongs to. The attribute `attr3` can be described by a comparator. Its descriptor can be a comparison with `attr2` or a comparison with a constant. This grammar allows rules like:

- if `attr1` = 0 and `attr2` between 50 180 and any, then `attr4` = T.
- if `attr1` = 2 and `attr2` is high and `attr3` ≠ 50, then `attr4` = T.
- if `attr1` = 1 and any and `attr3` >= `attr2`, then `attr4` = F.

The grammars for other problems are similar to the grammar in Table 4.1. According to the type of attribute, a descriptor similar to `Attr1_descriptor`, `Attr2_descriptor` or `Attr3_descriptor` can be used. The following list illustrates how the grammar is written for each situation.

- The attribute is nominal.

  The attribute can be described by its value. The descriptor similar to `Attr1_descriptor` or `Attr4_descriptor` can be used.

- The attribute is continuous.

  The attribute can be described by a range. The descriptor similar to `Attr2_descriptor` can be used.

- The attribute can be compared with other attributes in the rule

  In many case describing an attribute by a value is not powerful enough to represent the knowledge. If a comparison between variables is needed, the descriptor similar to `Attr3_descriptor` can be used.

- The attribute have more than one kind of descriptions.

  In some cases, an attribute can be described by more than one way. An example is `Attr2` in the previous example. By the use of grammar, we do not need to restrict the rule to use either one descriptor. Another example is that an address can be described by the city, state and country. This can be done by writing the grammar as follows:

  Address_descriptor → `Address between city_erc city_erc`

  Address_descriptor → `Address between state_erc state_erc`

  Address_descriptor → `Address between country_erc country_erc`

- The antecedent part have more than one format.

  The use of grammar allows the antecedents to have more than one format. As an example, the user may want that if `Attr1` is included in the antecedent, then `Attr3` and `Attr4` should also be included. Otherwise if `Attr2` is used instead of `Attr1`, then `Attr5` and `Attr6` should be included in the rule. This can be done by writing the grammar as follows:

  Antes → *Attr1* and *Attr3* and *Attr4*

  Antes → *Attr2* and *Attr5* and *Attr6*

	Rule
⇒	if *Antes* , then *Consq* .
⇒	if *Attr1* and *Attr2* and *Attr3* , then *Consq* .
⇒	if *Attr1_descriptor* and *Attr2_descriptor* and *Attr3_descriptor* , then *Attr4_descriptor* .
⇒	if attr1 = erc1 and attr2 between erc2 erc2 and attr3 *Comparator* *Attr3_term* , then attr4 = boolean_erc .
⇒	if attr1 = erc1 and attr2 between erc2 erc2 and attr3 ≠ erc3 , then attr4 = boolean_erc .
⇒	if attr1 = 0 and attr2 between 100 150 and attr3 ≠ 50 , then attr4 = T .

Table 4.2: An example derivation

- There are more than one target variable and thus more than one kind of rules.

  Usually data mining is not restricted to one target variable. The user may want to find knowledge describing all the dependent variables. Thus this leads to more than one kind of rules. Different kind of rules can be searched simultaneously in the search by starting the grammar as follows:

  Rule → *Rule1* | *Rule2*

  Rule1 → if *Antes1* , then *Consq1* .

  Rule1 → if *Antes2* , then *Consq2* .

4.2 Population Creation

The grammar is used to derive rules to make up the initial population. Each individual in the population corresponds to one rule. The *start symbol* is the first symbol of the first line of the grammar. From the start symbol, a complete derivation is performed. Every non-terminal is expanded according to the grammar until only terminals and ERCs are remained. If there are more than one possible derivation, a random choice is made. Table 4.2 illustrates how a rule is derived from the grammar in Table 4.1. The derivation tree of the derivation is stored as the genotype of the individual. The derivation tree for this derivation is shown in Figure 4.1.

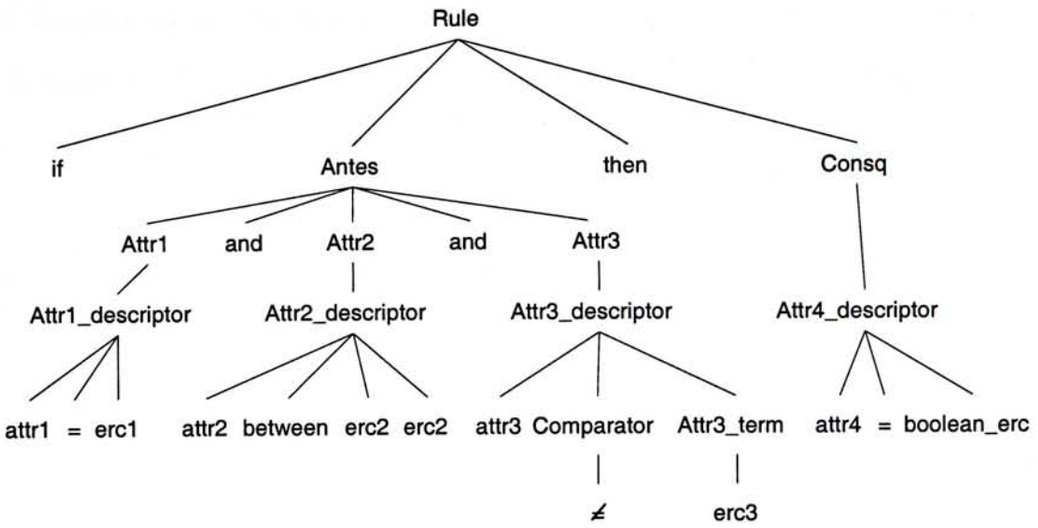


Figure 4.1: The derivation tree

After the derivation is completed, ERCs in the rules are instantiated. Our approach has two different ways to instantiate these constants. Conventional GP instantiates the constants randomly. A random value within the range of the ERC is assigned. Another way is to use *seeds* to generate better initial population. Using a seed can create a new rule that covers at least one record. When creating a new individual, a record in the training set is selected randomly as a seed. A rule is then derived from the grammar. During instantiating the ERCs, a constant is not generated randomly but generated to a value that matches the seed. For a nominal attribute, its ERC is instantiated to the value of the seed. For a continuous attribute that is described by a range, the ERCs are instantiated to a range that includes the value of the seed.

4.3 Genetic Operators

In rule learning using GGP, the search space is explored by generating new rules using three genetic operators: crossover, mutation and dropping condition. A rule is composed of attribute descriptors. The genetic operators try to change the descriptors in order to search for better rules.

Crossover is a sexual operation that produces one child from two parents. One

parent is designated as the primary parent and the other one as the secondary parent. A part of the primary parent is selected and replaced by another part from the secondary parent. Suppose that the following primary and secondary parents are selected:

if attr1=0 and attr2 between 100 150 and attr3≠50, then attr4=T.

if attr1=1 and any and attr3 >= attr2, then attr4=F.

The underlined parts are selected for crossover. The offspring will be

if attr1=0 and attr2 between 100 150 and attr3>=attr2, then attr4=T.

In GGP, each individual is represented by a derivation tree. The replaced part is actually a subtree selected randomly from the derivation tree of the primary parent (see Section 3.2). The subtree may represent different structures in the rule, hence the genetic change may occur either on the whole rule, on several descriptors, or on just one descriptor. The replacing part is also selected randomly from the derivation tree of the secondary parent, but under the constraint that the offspring produced must be valid according to the grammar. If a conjunction of descriptors is selected in the primary parent, it will be replaced by another conjunction of descriptors, but never by a single descriptor. If a descriptor is selected in the primary parent, then it can only be replaced by another descriptor of the same attribute. This can maintain the validity of the rule.

Mutation is an asexual operation. A part in the parental rule is selected and replaced by a randomly generated part (see Section 3.2). Similar to crossover, the selected part is a subtree of the derivation tree. The genetic change may occur on the whole rule, several descriptors, one descriptor, or the constants in the rule. The new part is generated by the same derivation mechanism as in the population creation. Because the offspring have to be valid according to the grammar, the selected part can only mutate to another part with a compatible structure. For example, the parent

if attr1=0 and attr2 between 100 150 and attr3≠50, then attr4=T.

may mutate to

if attr1=0 and attr2 between 100 150 and attr3>=attr2, then attr4=T.

Dropping condition is an genetic operator tailor-made for rule learning using GGP. Due to the probabilistic nature of GP, redundant constraints may be generated in the rule. For example, suppose that the actual knowledge is 'if A<20 then X=T'. We may learn rules like 'if A<20 and B<10 then X=T'. This rule is, of course, correct; but it is just a subsumed rule of the actual rule, and does not completely represent the actual knowledge. Dropping condition (Michalski [1983]) is incorporated in GGP to generalize rules. A rule can be generalized if one descriptor in the antecedent part is dropped. Dropping condition selects randomly one attribute descriptor, and then turns it into 'any'. That particular attribute is no longer considered in the rule, hence the rule can be generalized. For example, the parent

if `attr1`=0 and `attr2` between 100 150 and `attr3`$\neq$50, then `attr4`=T.

may change to

if `attr1`=0 and `attr2` between 100 150 and any, then `attr4`=T.

4.4 Evaluation of Rules

An evaluation function is needed to measure the degree of interesting of a rule. There are a lot of rule evaluation functions. Piatetsky-Shapiro [1991] suggested that for a rule 'if A then B', the function measuring the interesting of the rule should be a function of $p(A)$, $p(B)$, $p(A\&B)$, rule complexity and possibly other parameters (where $p(.)$ denotes the probability of .). Let N be the total number of training examples. Let $|A|$ denote the number of cases that satisfy a condition A, and $|A\&B|$ denote the number of cases that satisfy condition A and B, it is suggested that the rule-interest function RI should satisfy the following principles:

1. $RI = 0$ if $|A\&B| = \frac{|A||B|}{N}$. If A and B are statistically independent, the rule is not interesting.

2. RI monotonically increases with $|A\&B|$ when other parameters remain the same.

3. *RI* monotonically decreases with $|A|$ or $|B|$ when other parameters remain the same.

For a rule 'if A then B', the probability $p(A|B) = p(A\&B)/p(A)$ is the accuracy of the rule. According to the accuracy, a rule can be categorized as an *exact*, *strong* or *weak* rule. An exact rule is the rule that always correct, that is, $p(A|B) = 1$. A strong rule is a rule that almost always correct, that is, $p(A|B)$ is high. A weak rule is a rule that the occurrence of the consequent under the antecedent is much more than on average, that is $p(A|B) \gg p(B)$. In the real-life situation, an exact or strong rule may not exist. Thus a useful data mining system should not just search for exact or strong rules. It should be able to discover weak rules because the difference from average may already provide interesting knowledge. Consequently, accuracy cannot be the sole metric for rule-interest. Another measurement of rule-interest is the applicability of the rule to future cases. If the rule can match a larger number of training cases, it is less likely that the rule is just because of chance, and thus the rule should be more applicable to future cases.

An evaluation function based on the support-confidence framework (Agrawal et al. [1993]) is developed as the fitness function in our rule learning approach. *Support* measures the coverage of a rule. It is a ratio of the number of records covered by the rule to the total number of records. *Confidence factor* (cf) is the confidence of the consequent to be true under the antecedents, and is just the same as the rule accuracy. It is the ratio of the number of records matching both the consequent and the antecedents to the number of records matching only the antecedents. For a rule 'if A then B' and with a training set of N cases, support is $|A\&B|/N$ and confidence factor is $|A\&B|/|A|$.

In the evaluation process, each rule is checked with every record in the training set. Three statistics are counted. The number *antes_hit* is the number of records matching the antecedents (the 'if' part), *consq_hit* is the number of records that match the consequent (the 'then' part), and *both_hit* is the number of records that obey the whole rule (both the 'if' and the 'then' parts).

The confidence factor cf is the fraction *both_hit/antes_hit*. But a rule with a high confidence factor does not mean that it behaves significantly different from the average. Therefore we need to consider the average probability of the consequent (*prob*). The value *prob* is equal to *consq_hit/total*, where *total* is the total number of records in the training set. This value measures the confidence for the consequent under no particular antecedent.

A formula similar to the likelihood ratio used in CN2 (Equation 2.9) is used. We defined cf_part as

$$cf\_part = cf \times \log(\frac{cf}{prob}) \tag{4.1}$$

The log function measures the order of magnitude of the ratio $cf/prob$. This value is a product of two factors : cf and $\log(cf/prob)$. A high value of cf_part requires simultaneously a high value on the rule confidence (cf) and a high value on the rule confidence over the average probability ($cf/prob$). The definition of this value matches with the three previously stated principles proposed by Piatetsky-Shapiro [1991]. By using his notation, cf is actually $|A\&B|/|A|$,and *prob* is $|B|/N$. If $|A\&B| = |A||B|/N$, $cf/prob = 1$ and $cf\_part = 0$. The value cf (and so does cf_part) monotonically increases with $|A\&B|$ and monotonically decreases with $|A|$. The value *prob* monotonically increases with $|B|$ and thus cf_part monotonically decreases with $|B|$.

Support is another measure that we need to consider. A rule can have a high accuracy but the rule may be just because of chance and based on a few training examples. This kind of rules does not have enough support. The value *support* is defined as *both_hit/total*. If *support* is below a user-defined minimum threshold (*min_support*), the confidence factor of the rule should not be considered. This can avoid the waste of effort to evolve those rules with a high confidence but cannot be generalized.

We define our fitness function to be:

$$raw\_fitness = \begin{cases} support, & \text{if } support < min\_support \\ w_1 \times support + w_2 \times cf\_part, & \text{otherwise} \end{cases} \tag{4.2}$$

where the weights w_1 and w_2 are user-defined to control the balance between the confidence and the support in searching. We have set the values to 1 and 8 respectively so that the confidence of the rule plays a more important role in the evaluation function.

Chapter 5

Learning Multiple Rules from Data

The knowledge of a data set is unlikely to be sufficiently described by a single if-then rule. Multiple number of rules are required to represent the knowledge. To perform rule learning using evolutionary computation, a suitable modeling for individuals must be designed such that a set of rules can be learned. There are two different approaches. In the Pittsburgh approach (Smith [1980]; Smith [1983]), each individual in the population encodes a whole solution, that is, a set of rules. In the Michigan approach (Holland and Reitman [1978]; Booker et al. [1989]), each individual encodes only one rule. The individuals in the population can be combined together to provide a rule set. However this approach requires special techniques such that multiple good individuals can coexist in the population. Our approach uses the Michigan approach. The structure of an individual can be simpler because it only represents one rule. Thus the evolution for good individuals are easier.

This chapter begins with an review of previous approaches for maintaining groups of individuals evolving different solutions. Then our approach, token competition, is presented in Section 5.2. Section 5.3 summarizes the complete approach for rule learning. Experimental results of rule learning from two machine learning databases are presented in Section 5.4.

5.1 Previous approaches

Genetic algorithm and genetic programming are weak search algorithms to search for a solution that optimize the fitness function. These algorithms aim to search for a single solution only. Those individuals with higher fitness scores can survive while those with lower fitness scores will be extinct. If a part of the search space gives a higher fitness scores, eventually all the individuals will converge into this part.

However there are many situations that multiple solutions are required. For example, we may need to search for all the peaks in a multimodal function. In this case, it is desirable to maintain groups of individuals, with different groups evolving different solutions. Each group of individuals is referred as a sub-population or a species, and the part of the search space being explored by a species is referred as a niche. Maintaining diversity in the population is useful for the formulation of niches. The individuals are not allowed to converge to a single niche and hence forced to explore different part of the search space. Several approaches have been designed in GA to accomplish this task and they are reviewed in this section.

5.1.1 Preselection

Preselection (Cavicchio [1970]) maintains the diversity by trying to reduce the existences of similar individuals. It uses the idea that the parent should be one of the most similar individuals to the offspring. A new individual is evolved by using a genetic operator. The offspring can replace the parent if it has a better fitness. Otherwise the parents survives but not the child.

5.1.2 Crowding

In crowding (De Jong [1975]), a certain percentage of the population is selected to produce offspring. The percentage is denoted as the generation gap (G). Offspring are evolved by crossover and mutation to replace the original individuals in the population. To determine which individual is replaced, for each evolved offspring

several individuals are selected randomly from the population. The number of individuals selected is denoted as the crowding factor (CF). The similarity of the selected individuals with respect to the offspring is computed. Similarity is defined in turn of bit-wise (i.e. genotypic) matching. The individual that is the most similar to the offspring is replaced by the offspring.

5.1.3 Deterministic Crowding

Deterministic crowding (Mahfoud [1992]) improves preselection and crowding. In each generation, the individuals in the population are randomly paired without replacement. Each pair evolves two offspring by crossover. Deterministic crowding uses the idea of preselection that the offspring should be similar to its parent, and uses the idea of crowding that a similarity measure should used to determine the replacement. Deterministic crowding uses the phenotypic similarity. The bit strings of the individuals are decoded and the similarity measure is defined in the decoded parameters. The offspring are compared only with the two parents for similarity. There are two possible replacements of two parents by their two offspring: offspring 1 replaces parent 1 and offspring 2 replaces parent 2, or offspring 1 replaces parent 2 and offspring 2 replaces parent 1. The pair of replacements that yields the greatest sum of phenotypic similarities between offspring and the replaced parent is used. The parent is replaced by the offspring provided that the offspring can have a better fitness score.

5.1.4 Fitness sharing

Fitness sharing (Goldberg and Richardson [1987]) maintains a diversity of individuals by discouraging individuals to converge into one niche. The fitness of one individual gained from one niche must be shared by similar individuals. A distance function $d(x_i, x_j)$ measures the distance (i.e. dissimilarity) between two individuals x_i and x_j. For each individual, the distances with all other individuals are calculated. A sharing function s defines the degree of fitness sharing by the

similar individuals. The shared fitness f_s of one individual is the un-shared fitness f divided by the accumulated number of shares:

$$f_s(x_i) = \frac{f(x_i)}{\sum_j s(d(x_i, x_j))}$$

Thus when more individuals converge to one niche, the fitness is shared by more individuals. The fitness will decrease to a level such that it is no longer better than the fitness on other niches. Eventually a distribution of individuals on different niches can be achieved.

5.2 Token Competition

In our rule learning approach, the token competition (Leung et al. [1992]) technique is employed to increase the diversity, so that good individuals in different niches are maintained in the population. The concept is as follows: In the natural environment, once an individual has found a good place for living, it will try to exploit this niche and prevent other newcomers to share the resources, unless the newcomer is stronger than it is. The other individuals are hence forced to explore and find their own niches. In this way, the diversity of the population is increased.

Based on this mechanism, we assume each record in the training set can provide a resource called token. If a rule can match a record, it sets a flag to indicate the token is seized. Other weaker rules then cannot get the token. The priority of receiving tokens is determined by the strength of the rules. A rule with a high score on $raw\_fitness$ (Equation 4.2) can exploit the niche by seizing as many tokens as it can. The other rules entering the same niche will have their strength decreased because they cannot compete with the stronger rule. The fitness score of each individual is modified based on the tokens it can seize. The modified fitness is defined as :

$$modified\_fitness = raw\_fitness \times count/ideal \tag{5.1}$$

where $raw\_fitness$ is the fitness score obtained from the evaluation function, *count* is the number of tokens that the rule actually seized, *ideal* is the total number of tokens that it can seize, which is equal to the number of records that the rule matches. Token competition is a greedy operation. It favors strong rules as their chance of survival is maintained, while their close competitors are weakened as they cannot get the token.

From another point of view, each rule contributes to the system by covering several records of the database. If a record has already been covered by one rule, then another rule covering the same record will make no contribution to the system. Thus the fitness of the latter rule should be discounted.

Token competition is a simple method to force the diversity of the population. Token competition has an advantage that it does not require a distance function. In crowding or fitness sharing, it is required to define a similarity or a distance function, so as to measure the similarity or dissimilarity between two individuals. However, it may be difficult to define how one individual is similar to another individual, especially in Genetic Programming. Genetic Algorithm uses a fixed length binary string as the chromosome. Thus the genotypic difference (i.e. difference in the bits) can be used as a general similarity measurement. However this is not valid in the tree structure of Genetic Programming. Moreover, the similarity in genotype may not truly represent the similarity of the individuals. Token competition simplifies the problem by simply regarding two individuals to be similar if they cover the same record.

The execution of token competition is faster than fitness sharing. To calculate the fitness score of one individual in fitness sharing, the similarity scores of *all* other individuals with respect to this individual have to be calculated. If the similarity score can be computed in time t, and the population size is p, each individual needs a time pt to calculate the similarity score, and the time needed to complete fitness sharing in each generation is $O(p^2t)$. On the other hand, calculations of similarity are not needed in token competition. The required information

of token counting is the list of records that each individual covered. This information is already stored during the evaluation process. If an individual covers m records, a time of $O(m)$ is needed to seize the tokens, and token competition in each generation can be completed in $O(\bar{m}p)$, where $\bar{m}$ is average value of m. This computation is straight forward and can be faster than fitness sharing if $O(\bar{m}) < O(pt)$.

As a result of token competition, there are rules that cannot seize any token. These rules are redundant as all of their records are already covered by the stronger rules. They can be replaced by new individuals. Introducing these new individuals can inject a larger degree of diversity into the population, and provide extra chances for generating good rules. To create the new individuals, we can use seeds to generate better rules (see Section 4.2). Those records with their tokens not taken are the possible seeds. These records are not yet covered by any existing rules, and thus introducing rules covering them can improve the system. To create a new rule, a seed is selected, and then the rule is generated to cover the seed. The constants in the rule will not be instantiated randomly but with values matching with the seed.

5.3 The Complete Rule Learning Approach

Figure 5.1 is the flowchart of the complete process for learning multiple rules from a set of data using GGP. A grammar is provided by the user as a template for rules. A set of rules is derived by using this grammar and forms the initial population. Then, the main loop of GGP is entered. In each generation, individuals are selected stochastically to evolve offspring by the three genetic operators: crossover, mutation and dropping condition. In each generation, the number of new individuals evolved equals to the population size. Thus at this stage, the number of individuals in the population is doubled. All individuals participate in the token competition and the replacement step, so as to eliminate similar rules and increase the diversity. One half of the individuals with the higher fitness

scores after token competition are retained and passed to the next generation. The whole process iterated until the maximum number of generations has been reached.

Parents for the genetic operators are selected by the rank selection method (see Section 3.1.5). The probabilities of using crossover, mutation and dropping condition in our approach are 0.5, 0.4 and 0.1 respectively. These settings are chosen because they gave the best result in preliminary executions of the system.

The data set for learning can be partitioned into a training set and a testing set. Only the training set is available for the learning process. After the maximum number of generations is reached, the discovered rules are further evaluated with the unseen testing set, so as to verify their accuracy and reject the rules that over-fit the training set.

Our system differs from conventional GP that reproduction operator is not used, and the parents compete with the offspring for places in the new generation. In conventional GP, the next generation of population only consists of the offspring. An individual will be passed to the next generation of population through the use of the reproduction operator. Good individuals can exploit their genes to the new generation by reproducing more children, and gradually dominate the population. Thus many individuals contain the good genes, and a good gene has a high probability of being passed to the offspring. However, in our rule learning approach, we do not want a good rule to replicate itself and dominate the population. Rather, we need to find several good rules and diversify the population. Token competition only allows one copy of each good individual to be kept in the population. Consequently, the chance of a good gene being passed to the offspring is much less than conventional GP, because a good individual may not be selected as the parent. Therefore we need an explicit way to retain the good genes of the parents. This is done by keeping the parents as competitors for the new generation. Good parents can win poor offspring and gain positions in the new generation.

The execution time can be approximated by assuming that the evaluation of

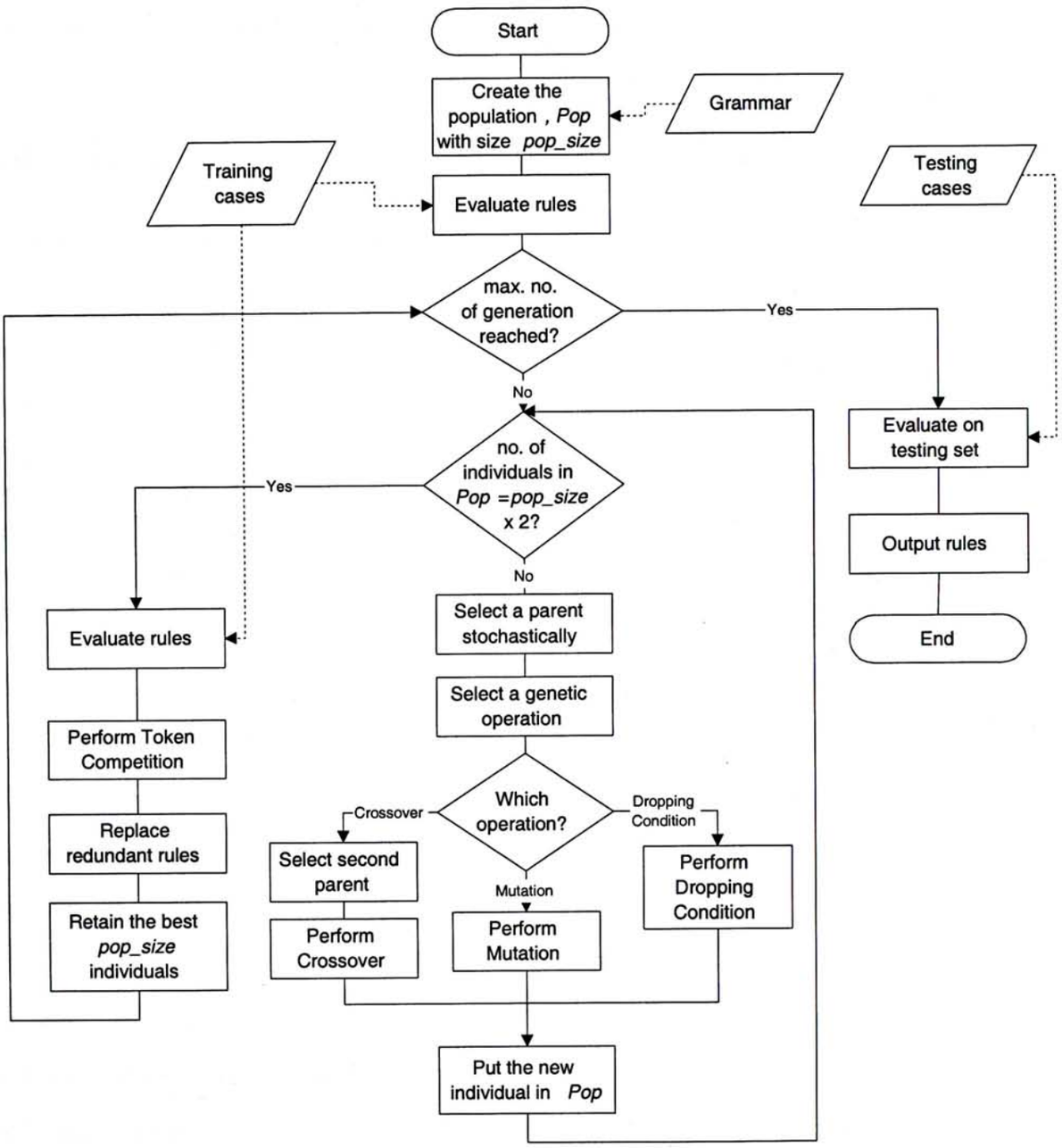


Figure 5.1: The flowchart of the Rule Learning process

rules is the most time consuming step. In each generation, each rule has to be checked with every training case to count the number of records that match the antecedents or the consequent. Thus we can roughly estimated that the execution time should be directly proportional to (number of database records) $\times$ (population size) $\times$ (number of generations).

5.4 Experiments with Machine Learning Databases

Experiments have been performed to evaluate the rule learning system. Two databases from the UCI Machine Learning Repository (Merz and Murphy [1998]) are used as the source of data. In these database the target is to search for knowledge for classification. A useful measure of the accuracy of the learned knowledge is to apply it to an unseen testing set. Thus the database is divided into a training set and a testing set. To measure the accuracy in the testing set, the rules are applied to see whether each testing case is classified correctly. Since the discovered rules can overlap, a testing case may match more than one rule. Starting from the rule with the highest fitness value, the testing case is checked by each rule. If the antecedent part does not match with the testing case, the next rule is applied until there is a match or no rule can apply. If no rule can be applied or the testing case matches the antecedents but not the consequent part, then the testing case is considered as a miss.

We should note that the classification accuracy on testing sets is different from the rule accuracy. For a rule with a high rule accuracy, the classification accuracies on those cases that match the antecedent part will be high. However the rules with high accuracies may not cover all the testing cases. It is possible that a testing case only matches with a less accurate rule, and the overall classification accuracy will then be lower. The aim of our rule learning approach is to discover knowledge instead of classifying unseen cases. No special technique is designed to make the rules cover all the cases. Thus the classification accuracy is only an indirect measurement of our approach.

Attribute	Type	Possible Value
sepal length (in cm)	continuous	4.3-7.9
sepal width (in cm)	continuous	2.0-4.4
petal length	continuous	1.0-6.9
petal width	continuous	0.1-2.5
class	nominal	Iris setosa, Iris Vericolor, Iris Virginica

Table 5.1: The iris plants database

```
Rule → if Antes , then Consq .
Antes → slength and swidth and plength and pwidth
slength → any | slength_descriptor
swidth → any | swidth_descriptor
plength → any | plength_descriptor
pwidth → any | pwidth_descriptor
slength_descriptor → sepal_length between slength_const slength_const
swidth_descriptor → sepal_width between swidth_const swidth_const
plength_descriptor → petal_length between plength_const plength_const
pwidth_descriptor → petal_width between pwidth_const pwidth_const
Consq → class_descriptor
class_descriptor → class is class_const
```

Table 5.2: The grammar for the iris plants database

5.4.1 Experimental results on the Iris Plant Database

The first experiment uses the iris plants database as the data set. This database is one of the most frequently used database in machine learning. It consists of 150 records with 5 attributes (Table 5.1). The task is to discover knowledge about the three classes. Each class has 50 records in the database. 100 records are randomly selected as the training set and the remaining 50 records are used as the testing set.

The grammar in Table 5.2 is used for learning rules from this database. This grammar is very simple. Each of the four continuous attributes is described by a range in the rule, and the nominal attribute is described by a value. This grammar is used to create a population with size 50. The maximum number of generations is 50.

Preliminary experiments are performed to investigate the effects of different parameter settings. We found that by lowering the value of w_2 in the fitness

w_1	w_2	Accuracy
1	8	87.6%
1	1	91.6%

Table 5.3: Results of different value of w_2

Minimum support	Accuracy
0.03	91.6%
0.01	94.8%

Table 5.4: Results of different value of minimum support

function (Equation 4.2), a higher accuracy on the testing set can be achieved, as shown in Table 5.3. In this database it is quite easy to find a rule with a high confidence, but the rule may not be general enough. Since the rule set needs to cover all testing cases, the goal of the evolution process is not just to evolve rules with high confidence, but also to evolve rules with high support. A lower value of w_2 in the fitness function can favor more general rules with a better support. We also found that the classification accuracy on using a lower value of minimum support is somewhat better, and the result is less sensitive to the rates of the genetic operators. The results are shown in Table 5.4 and 5.5.

A more complete result is obtained by executing 25 runs using the best setting that we have tried. The best setting uses a rate of 0.5 for crossover, 0.4 for mutation, and 0.1 for dropping condition, 0.01 for minimum support, 1 and 1 respectively for the values of w_1 and w_2 for the fitness function. The execution time for each run is about 70 seconds in a Sun Ultra 1/140. Our system gets an average classification accuracy of 91.04%. The results of these runs are shown in Table 5.6. The best run gives an accuracy of 100% and the rules are listed in

Rate of			
Crossover	Mutation	Dropping condition	Accuracy
0.50	0.40	0.1	94.8%
0.35	0.55	0.1	92.4%
0.60	0.30	0.1	91.6%
0.45	0.35	0.2	94.8%

Table 5.5: Results of different probabilities for the genetic operators

Accuracy				Time
Mean	Standard Derivation	Maximum	Minimum	
0.9104	0.0548	1.00	0.76	70 sec.

Table 5.6: Experimental result on the iris plants database

Approach	Accuracy
Our approach	91.04% (100%)
C4.5	93.8%
ID3	94.2%
Nearest Neighbor	96.0%
Neural Net	96.7%

Table 5.7: The classification accuracy of different approaches on the iris plants database

Appendix A.1.

The results of other approaches are quoted from Holte [1993] as references (Table 5.7). It should be notice that these results are obtained using different number of runs and different settings in the training and testing set. For our approach, the value inside the brackets shows the best accuracy. The best accuracies of the other approaches are not available. However, as they are deterministic approaches, their accuracies do not vary on different runs. Their results are different only because the training and testing sets used are different. The average accuracy of our approach shown in this table is not as good as the other approaches. However, the perfect result can be obtained in the best run of our approach. A characteristic of evolutionary algorithms is that they are stochastic. Thus our approach has larger fluctuations in different runs and it is improper to compare the average accuracy of our approach to other approaches. In order to get a better result, the user may execute several trials of the algorithm to get the result with the best fitness score.

5.4.2 Experimental results on the Monk Database

The second experiment is performed on the Monk database (Thrun et al. [1991]). This database contains attributes for artificial robots, as shown in Table 5.8. There are three data sets. Each data set has a hidden knowledge on the robots

Attribute	Possible Value
head shape	1(round), 2(square), 3(octagon)
body shape	1(round), 2(square), 3(octagon)
is smiling	1(yes), 2(no)
holding	1(sword), 2(balloon), 3(flag)
jacket color	1(red), 2(yellow), 3(green), 4(blue)
has tie	1(yes), 2(no)
class	1(yes), 2(no)

Table 5.8: The monk database

that belong to the class (i.e. class = 1). The training set contains randomly selected robots while the testing set contains all the 432 possible robots. The task is to discover the knowledge on classification of a robot into the positive or negative class.

1. The monk1 data set has 124 examples in the training set, which contains 62 positive examples (i.e. class=1) and 62 negative examples (i.e. class=2). The testing set contains 216 positive and 216 negative examples. The hidden knowledge for classification is "(head shape = body shape) or (jacket color = 1)". There were no mis-classifications.

2. The monk2 data set has 169 examples in the training set, which contains 105 positive and 64 negative examples. The testing set contains 190 positive and 142 negative examples. The knowledge hidden is "exactly two of the six attributes have the values 1". For example, a robot with head shape=1, body shape=3, is smiling=1, holding=3, holding=2 and jacket color=2 is positive. There were no mis-classifications.

3. The monk3 data set has 122 examples in the training set, which contains 62 positive and 60 negative examples. The testing set contains 204 positive and 228 negative examples. The knowledge hidden is "(holding = 1 and jacket color = 3) or (body shape $\neq$ 3 and jacket color $\neq$ 4). There were 5% mis-classifications in the training set.

Rule → if *Antes* , then *Consq* .
Antes → *shape1* and *smile1* and *hold1* and *jacket1* and *tie1*
shape1 → *shape_comparison* | *head1* and *body1*
shape_comparison → head_shape *comparator* body_shape
head1 → any | *head_descriptor*
body1 → any | *body_descriptor*
smile1 → any | *smile_descriptor*
hold1 → any | *hold_descriptor*
jacket1 → any | *jacket_descriptor*
tie1 → any | *tie_descriptor*
head_descriptor → head_shape *comparator* erc3
body_descriptor → body_shape *comparator* erc3
smile_descriptor → is_smiling *comparator* erc2
hold_descriptor → holding *comparator* erc3
jacket_descriptor → jacket_color *comparator* erc4
tie_descriptor → has_tie *comparator* erc2
comparator → = | ≠
Consq → positive

Table 5.9: The grammar for the monk database

The knowledge in monk1 is in the standard disjunctive normal form (DNF). The knowledge in monk2 is similar to a parity problem, and is difficult to be described in DNF using the given attributes only. The knowledge in monk3 is again in DNF but under the presence of noise.

The grammar for learning rules from this database is listed in Table 5.9. In this database, there should be only one kind of rule: rules describing knowledge about the positive robot. Thus the rules can only have one consequent: 'positive'. To classify a case as negative, a default rule 'if any then negative' is used. The fitness of this rule is calculated. A discovered rule is not used if its fitness is below the default. If no rule can be applied to a case, then the default rule is used. In this grammar, the attributes head shape and body shape can be described in two ways. Basically each attribute can be described by its value. However as they are both about the shape, a possible description is a comparison of them. The other attributes are described by their values. The constants `erc2`, `erc3` and `erc4` are respectively with the range 1 to 2, 1 to 3 and 1 to 4.

	Accuracy			
	Mean	Standard Derivation	Maximum	Minimum
Monk1	1.000	0.000	1.00	1.00
Monk2	0.600	0.091	0.69	0.31
Monk3	0.954	0.029	1.00	0.89

Table 5.10: Experimental result on the Monk database

Approach	Monk1	Monk2	Monk3
ID3	98.6%	67.9%	94.4%
AQR	95.9%	79.7%	87.0%
CN2	**100%**	69.0%	89.1%
AQ17-DCI	**100%**	**100%**	94.2%
AQ15-GA	**100%**	86.8%	**100%**
Assistant Professional	**100%**	81.3%	**100%**
Backpropagation	**100%**	**100%**	93.1%
Our approach	**100% (100%)**	60% (69%)	95.4% (**100%**)

Table 5.11: The classification accuracy of different approaches on the monk database

For each data set, rule learning is executed for 25 runs using the following setting: population size is 50, maximum number of generations is 50, the rates for crossover, mutation and dropping condition are 0.5, 0.4 and 0.1 respectively, minimum support is 0.01, w_1 is 1 and w_2 is 8. The execution time for each run is around 120 seconds. The result is shown in Table 5.10. The average results of other approaches are quoted from Thrun et al. [1991] in Table 5.11 as references. The best results of our approach are shown inside the brackets.

- Monk1 database

  For the monk1 database, the hidden knowledge can be easily reconstructed by the above grammar. Thus we can obtain a 100% classification accuracy on each run. The rule set is shown in Appendix A.2.1. If the grammar does not include a comparison between head shape and body shape, the perfect rule set can still be found but at a later generation, and three rules are needed to represented the concept (head shape = body shape) using the three possible values.

- Monk2 database

  The hidden knowledge is difficult to be represented using a context free grammar. The simple hidden rule must be represented by a large number of rules. Our system cannot evolve all of these rules and results in a poorer classification accuracy. Rules with this simply format have limited knowledge representation power, and cannot represent a certain kind of knowledge. Approaches that does not use simply rules, such as the backpropagation neural network, can achieve a much better result.

  The result of our approach may be improved if evolution using a context *sensitive* grammar is implemented in the system. The best rule set is shown in Appendix A.2.2.

- Monk3 database

  Our system can discover knowledge with a high classification accuracy under this noisy environment. The accuracy is the third best in these approaches, and the best rule set, shown in Appendix A.2.3, can classify all testing cases correctly.

From these experiments, we can see that our rule learning approach can successfully learn rules with high accuracy from the data, although the perfect rule set may not be discovered in every run.

Chapter 6

Bayesian Network Learning

In the approach of rule learning, we have focused on the detail view of the data. One deficiency of this approach is that the discovered rule set is not guaranteed to cover the whole database. The system will not provide any knowledge for a record that is not covered by any rules. The rules can describe parts of the database that have interesting patterns, but do not provide a general knowledge on the data. Moreover, the rules in the rule set are not organized. A causality relationship may be expanded into several similar rules. The rule learning step is not able to organize them into a chain and cannot provide a generalized view.

A Bayesian network can be a complement to rules. A Bayesian network is a much different model to represent the knowledge of data. It captures the conditional probabilities between variables (i.e. attributes in the database), and focuses on the general relationships between variables. In many real-life situation, the data just cannot be described completely by a few rules. Building a complete model for such a database is difficult and usually results in a complicated model. Bayesian network should be a suitable knowledge representation to give a structural causality model. It is easy to understand because of its graphical representation, while it has a well-developed mathematical model and can be used to perform reasoning under uncertainty.

Wong et al. [1997] has introduced an approach based on the Minimum Description Length Principle (MDL) and Evolutionary Programming (EP) to learn

Bayesian networks. However, the learning of Bayesian network is limited to discrete variables only. As Friedman and Goldszmidt [1996] has extended the definition of MDL score to handle continuous variables (Section 2.5.2), it is possible to combine these two researches, such that evolutionary computation can be used to learn a Bayesian network from a data set with discrete as well as continuous variables.

The approach MDLEP, which uses EP to optimize the MDL score, is introduced in Section 6.1. Then this approach is extended by introducing another layer to learn a discretization policy to discretize the continuous variables. This new layer uses Genetic Algorithm as the search method, and is described in Section 6.2. The experimental results of the new combined approach are given in Section 6.3.

6.1 The MDLEP Learning Approach

The approach MDLEP (Wong et al. [1997]; Lam et al. [1998]) uses EP to optimize the MDL metric (Equation 2.14), so as to learn the best Bayesian network. The flowchart in Figure 6.1 shows the process. Each individual represents a network structure, which is a directed acyclic graph (DAG). A connection matrix is used to represent the graph. A set of individuals is randomly created to make up the initial population. Each graph is evaluated by the MDL metric. Then each individual produces a child by performing a number of mutations. The child is also evaluated by the MDL metric. The next generation of population is selected among the parents and children by tournaments. Each DAG B is compared with q other randomly selected DAGs. The tournament score of B equals to the number of rivals that B can win, that is, the number of DAGs among those selected that have higher MDL scores than B. In our setting, the value of q is 5. One half of DAGs with the highest tournament scores are retained for the next generation. The process is repeated until the maximum number of generations is reached. The setting on the maximum number of generations depends on the complexity

of the network structure. If we expect a simple network, the maximum number of generations can be set to a lower value. The network with the lowest MDL score is output as the result.

Offspring in EP is produced by using a number of mutations. The probabilities of using 1, 2, 3, 4, 5 or 6 mutations are set to 0.2, 0.2, 0.2, 0.2, 0.1 and 0.1 respectively. The mutation operators modify the edges of the DAG. If a cyclic graph is formed after the mutation, edges in the cycles are removed to keep it acyclic. The approach uses four mutation operators, with the same probabilities of being used:

1. Simple mutation randomly adds an edge between two nodes or randomly deletes an existing edge from the parent.

2. Reversion mutation randomly selects an existing edge and reverses its direction.

3. Move mutation randomly selects an existing edge. It moves the parent of the edge to another node, or moves the child of the edge to another node.

4. Knowledge-Guided mutation is similar to simple mutation, but the MDL scores of the edges guide the selection of the edge to be added or removed. The MDL metric of all possible edges in the network is computed before the learning algorithm starts. This mutation operator stochastically adds an edge with a small MDL metric to the parental network or deletes an existing edge with a large MDL metric.

6.2 Learning of Discretization Policy by Genetic Algorithm

Friedman and Goldszmidt [1996] have extended the definition of MDL to include the discretization of continuous attributes (see Section 2.5.2). However the search

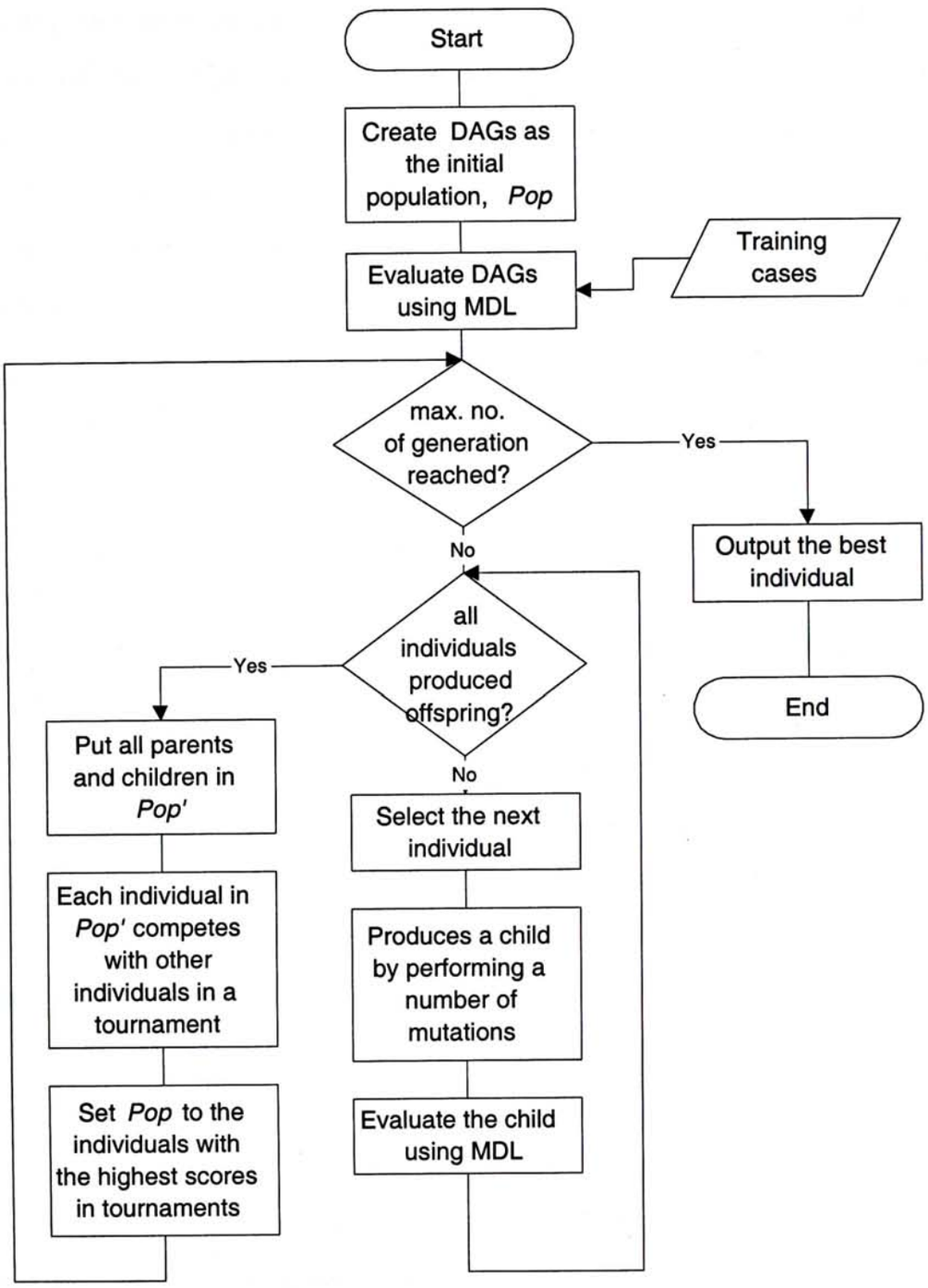


Figure 6.1: The flowchart of the MDLEP process

algorithm they proposed has a serious deficiency: The algorithm is a greedy approach and can be easily trapped in a local optima with no way to escape. This approach also greatly depends on the initial settings. If the initial guess of discretization policy or network structure is not good, the result can be poor.

An evolutionary approach can be applied to optimize the new MDL metric, and thus the best network structure as well as the best discretization policy can be learned. The use of evolutionary computation can have less chance for being trapped in a local optima, because there is a population of individuals to explore the search space in parallel. However, the search space is very huge, since the optimization includes two aspects: the optimizations of the network structure as well as the discretization policy. There are also two different kinds of genetic changes: genetic changes in the DAG and genetic changes in the discretization policy. Thus optimizing both aspects in one step is difficult and inappropriate.

A more realistic approach is to use the iterative approach as suggested by Friedman and Goldszmidt [1996]. In both the learning of the network structure and the discretization policy, evolutionary approach can be used. MDLEP can be applied directly to the network learning step. On the learning of the discretization policy, we have applied Genetic Algorithm as the search algorithm. Thus, started with an initial discretization policy, MDLEP is used to learn the network structure. Based on this structure, GA is used to learn the discretization policy. The process is iterated until the maximum number of iterations is reached.

The genetic algorithm starts with an initial randomly generated population. Each individual in the population is evaluated by the new MDL score defined in Equation 2.18. The good individuals are selected to produce offspring using the genetic operators. The offspring in turn produces the next generation until the maximum number of generations is reached.

6.2.1 Individual Representation

In this problem we want to search for a good discretization policy. A discretization policy consists of discretization sequences for the continuous variables, and each

Values of variable a :	1	2	3	5	8	12	15	20	33	40
Bit string of variable a :	0	1	0	0	0	1	0	0	1	

Figure 6.2: A bit string represents a discretization sequence

0	1	0	0	0	1	0	0	1	1	0	0	1	0	1	0	0	···
variable a									variable b								···

Figure 6.3: The bit string in an individual

discretization sequences consists of threshold values for discretization. We can limit the thresholds to mid-points between successive values that appeared in the training data. Each individual should represent a possible discretization policy, and hence each individual should encode these threshold values.

We have used one bit string to represent one discretization sequence. The number of bits in each string equals to the number of mid-points values of the variable (i.e. if variable i has s_i different values in the training data, the length of its bit string is $s_i - 1$). A '1' in the bit means the corresponding mid-point is included as a threshold in the discretization sequence. For example, if variable a has 10 different values, its values appeared in the data set and the corresponding bit string are as shown in Figure 6.2, then variable a is discretized into four values: 1-2 are discretized to a value 1, 3-12 are discretized to 2, 15-33 are discretized to 3 and 40 is discretized to 4. The thresholds represented in this bit string are the mid-points between the successive values, i.e. 2.5 (mid-point of 2 and 3), 13.5 (mid-point of 12 and 15) and 36.5 (mid-point of 33 and 40). To provide a more useful discretization and simplify the computation, the user can limit the maximum number of thresholds appeared in the discretization sequence. Hence the maximum number of '1' in the bit string is limited. An individual stores the concatenation of the bit strings of each continuous variable, as shown in Figure 6.3.

6.2.2 Genetic Operators

Four genetic operators are used. Other than the basic operators of reproduction, crossover and mutation, another operator named 'shift' is applied to evolve better discretization policies:

- Reproduction: The standard reproduction is used. The parent is selected and copied into the new generation.

- Crossover: The standard crossover can also be used. Two parents are selected. One random point in the bit string of the parents is selected as the crossover point. The bit string is cut into two parts at this point. The upper parts of the two parents are exchanged to evolve two children. Then the number of '1's for each continuous variable is counted. If the number of thresholds for a variable is larger then the limit, one or more '1's in the bit string are randomly selected and turned into '0'.

- Mutation: The mutation we used is a multiple-point mutation. A parent is selected. A random bit from each variable is selected for mutation. In a special case that the limit of '1's is already reached, only '0's in the bit string are selected for mutation. There is a 50% chance that the bit is mutated. If mutation occurs, the the selected bit is changed from 0 to 1 or vice versa.

- Shift: Shift is a special kind of mutation. A slightly change of the threshold values would not change the effect of the discretization greatly. Thus a slightly increases or decreases of a threshold that gives a good fitness score will give a good or hopefully even better fitness score. One parent is selected for the shift operator. A random bit with '1' is selected from each variable. For each bit there is a 50% chance that a threshold value is shifted. If shift occurs, the bit is set to '0' and its neighbor bit (either left or right, with equal probabilities) is set to '1'. This effectively changes the threshold value in the discretization sequence to the next (or previous) mid-point value.

6.3 Experimental Results

The performance of the GA approach is evaluated on a machine learning database and two artificially generated databases. A network structure as well as a discretization policy are learned from the data. The network structure and discretization policy is searched alternatively. MDLEP is used as the algorithm for network structure learning. We have used a population size of 50 to run for 75 generations in MDLEP. In the GA approach, we used a population size of 50 to run for 50 generations. The probabilities of using reproduction, crossover, mutation and shift are 0.2, 0.4, 0.2 and 0.2 respectively.

Each continuous variable is initially discretized to two values by a single random threshold value. A variable can at most have 5 discretization thresholds (i.e. a maximum of 6 ranges). The learning of network structure and discretization policy are alternated for 20 iterations. At each iteration of learning the network structure, MDLEP is re-started from scratch. However at each iteration of learning the discretization policy, an elitism is employed. The best discretization policy learned from the previous iteration of GA is retained as one individual in the population, and the other individuals are created randomly.

We also compare the result with the greedy approach. The greedy approach described in Section 2.5.2 is implemented. MDLEP with the same setting is used to learn the network structure. Since the greedy approach is quite sensitive to the initial setting, two different initial discretizations have been tested. In Greedy0 the initial discretization is the same as the GA approach. In Greedy1 the discretization threshold is set to the median of the set of possible values. For example, if the continuous variable has n different values, the 1^{st} to the $\lfloor n/2 \rfloor^{th}$ value are initially discretized to one value and the remaining values are discretized to another value. Although Greedy1 starts with a fixed discretization, MDLEP will give a different result in each run, and thus the results of different runs of Greedy1 will be different.

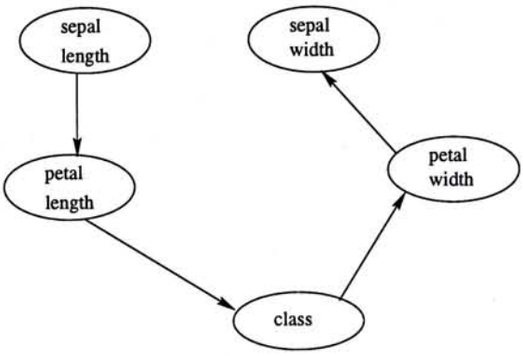
	mean	min.	max.	S.D.
GA	2574.81	2506.08	2759.77	75.81
Greedy0	2693.93	2574.68	2952.72	157.41
Greedy1	2574.68	2574.68	2574.68	0.00

Table 6.1: Results of experiment 1

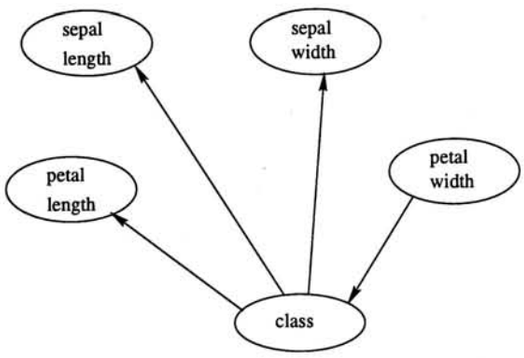
6.3.1 Experiment 1

In the first experiment, a Bayesian network is learned from the Iris plants data set described in Section 5.4.1. Each approach is executed for 10 trials. Their mean, minimum, maximum and standard derivation of the MDL score of these trials are shown in Table 6.1. It shows that the average score of GA is equally good as Greedy1, and better than Greedy0. For the best trial (i.e. with the minimum score), GA can give a better score than the other two approaches. The network structure and the discretization policy of the best trial of these approaches are respectively shown in Figure 6.4 and Figure 6.5.

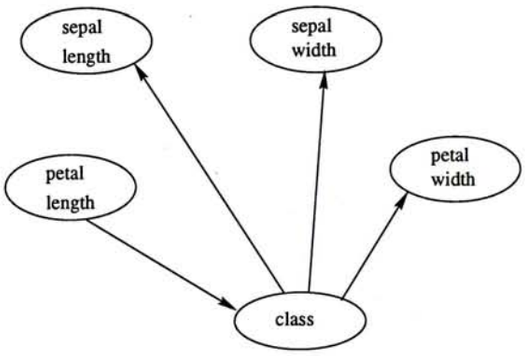
The three different approaches give different network structures but similar discretization policy in the best trial. There is no evidence to say whether these network structure and discretization policy is correct. However, the distribution of values and the rule discovered by the rule learning approach (shown in Appendix A.1) can give suggestions. In the database, petal length does not have records with values between 2.0 and 2.9, and petal width does not have records with values between 0.7 and 0.9. This suggests that petal length smaller than 2.0 should belong to one group and larger than 2.9 should belong to another, and petal width smaller than 0.7 should belong to one group and larger than 0.9 should belong to another. The results of all approaches can achieve this. Meanwhile, the first rule of Appendix A.1 shows that petal width between 0.0 to 0.8 can imply a class of iris-setosa. Thus it is reasonable to discretize values in this range to one value. All approaches can successfully achieve this. The second rule shows that petal length between 2.0 and 5.0, and petal width between 0.2 and 1.7 can imply a class of iris-versicolor. Thus it is reasonably to discretize petal length between 2.0 and 5.0 to one value, and petal width between 0.2 and 1.7 to one value. All



(a) GA



(b) Greedy0



(c) Greedy1

Figure 6.4: The network structures of the best results of Experiment 1

sepal length: [4.3-5.4] [5.5-5.8] [5.9-7.9]
sepal width: [2-2.9] [3-3.3] [3.4-4.4]
petal length: [1-1.9] [3-4.7] [4.8-6.9]
petal width: [0.1-0.6] [1-1.7] [1.8-2.5]

(a) GA

sepal length: [4.3-5.5] [5.6-6.1] [6.2-7.9]
sepal width: [2-2.9] [3-3.3] [3.4-4.4]
petal length: [1-1.9] [3-4.7] [4.8-6.9]
petal width: [0.1-0.6] [1-1.7] [1.8-2.5]

(b) Greedy0

sepal length: [4.3-5.5] [5.6-6.1] [6.2-7.9]
sepal width: [2-2.9] [3-3.3] [3.4-4.4]
petal length: [1-1.9] [3-4.7] [4.8-6.9]
petal width: [0.1-0.6] [1-1.7] [1.8-2.5]

(c) Greedy1

Figure 6.5: The discretization policies of the best results of Experiment 1

approaches give a discretization for petal length between 3.0 and 4.7. Petal width is not discretized by the range [0.2-1.7] in the best results, but by two ranges [0.1-0.6] [1-1.7]. An extra range is necessary as suggested by the distribution of values and by the first rule. These observations suggested that appropriate discretization policies can be successfully constructed by all approaches.

6.3.2 Experiment 2

In the second experiment, a simple Bayesian network structure is used to generate a data set of 1000 records, as shown in Figure 6.7(a). Variable 0, 1, 2, 4, 5 are independent variables and their values are distributed differently: Variable 0 is normally distributed in the range (0-0.3), (0.3-0.6) and (0.7-1.0); Variable 1 is normally distributed in the range (0-0.4) and (0.6-1.0); Variable 2 is normally distributed in the range (0.1-0.6) and (0.4-1.0); Variable 4 is normally distributed in the range (0-0.2), (0.4-0.8) and (0.8-1.0); Variable 5 is *uniformly* distributed on (0-1.0). The actual frequency distributions of values in the data set are shown in

	mean	min.	max.	S.D.
GA	39604.1	39435	40205	234.37
Greedy0	41040.1	40408	41603	410.87
Greedy1	40291.1	40236	40403	48.64

Table 6.2: Results of experiment 2

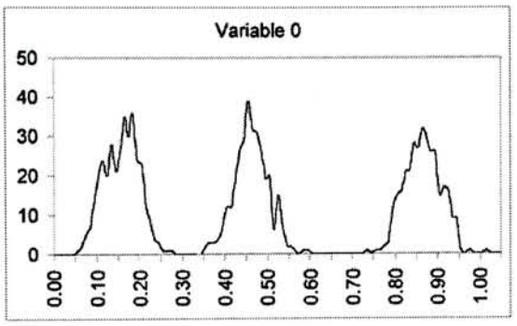
Figure 6.6.

The performance of ten trials of each approach is shown in Table 6.2. GA is better than the other two approaches both on the average score and the best score. The network structures of the best result of each approach as well as the original structure are shown in Figure 6.7. Both approaches cannot reconstruct the original structure, but the network constructed by GA is the most similar to the original one. It should be noted that MDL score gives a trade off between simplicity and accuracy. Thus the original structure may not be the structure with the best MDL score.

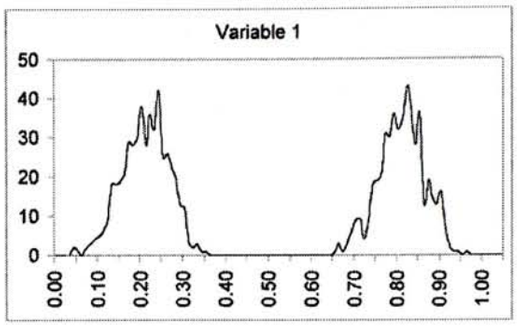
The discretization policies of the best results are shown in Figure 6.8. Both GA and Greedy1 can discretize variable 0,1,2 and 4 according to the given distributions, while Greedy0 failed to discretize variable 0. For the other variables, the discretizations generally match the fluctuations in the frequency distribution. Figure 6.9 is an example showing how variable 3 is divided into 6 ranges by the discretization policy of GA. The frequency distribution can show the probability $p(X_i|X_i^*)$, which affects the encoding length for reconstruction (see Section 2.5.2 and Equation 2.20). Nevertheless, the encoding length for reconstruction is only one part of the MDL score. A good discretization policy should optimize this part as well as the other parts.

6.3.3 Experiment 3

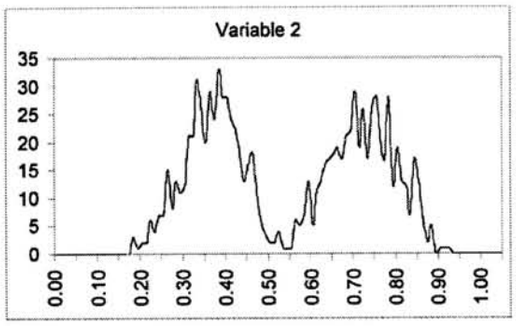
In the third experiment, a more complex structure (Figure 6.11(a)) is used to generate 1000 data. The independent variables are variable 2, 5, 6 and 7. Variable 2 is normally distributed in the range (0-0.4), (0.4-0.7) and (0.75-1.0); Variable 5 is normally distributed in the range (0-0.4) and (0.6-1.0); Variable 6 is normally



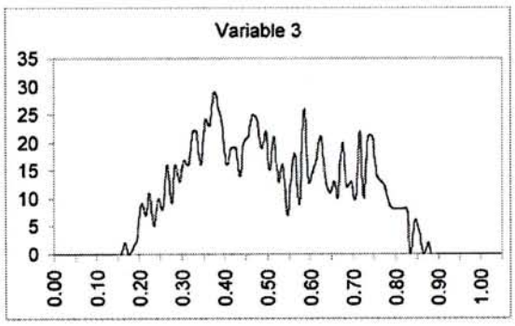
(a) Variable 0



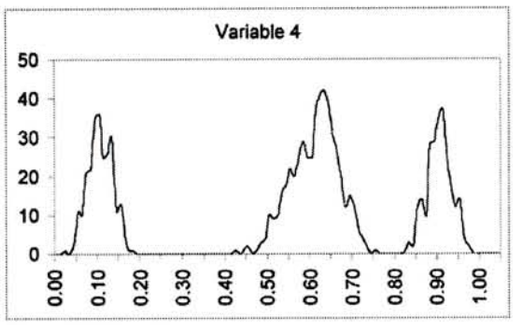
(b) Variable 1



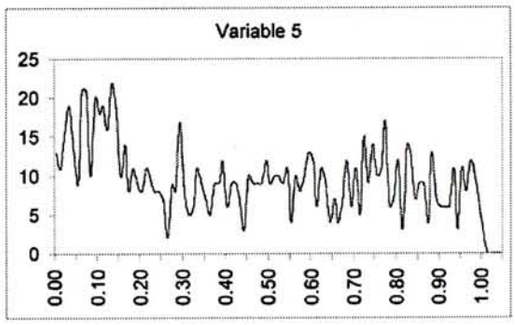
(c) Variable 2



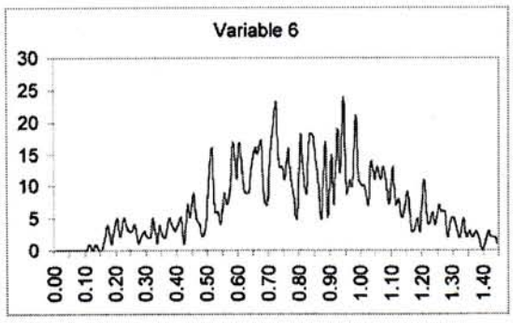
(d) Variable 3



(e) Variable 4

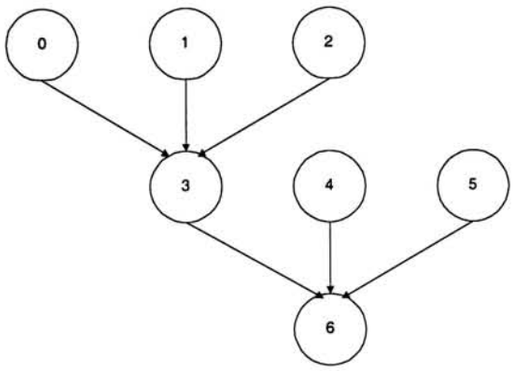


(f) Variable 5

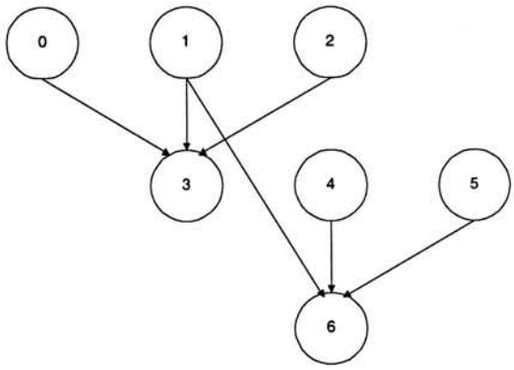


(g) Variable 6

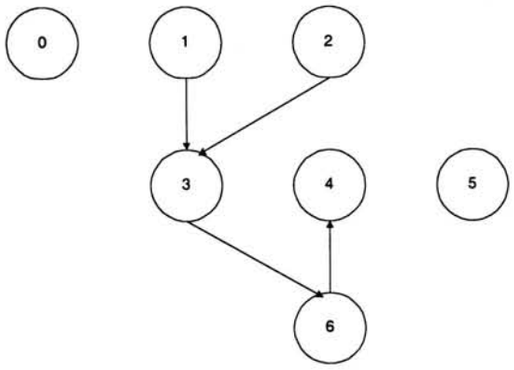
Figure 6.6: The frequency distribution of the variables of experiment 2



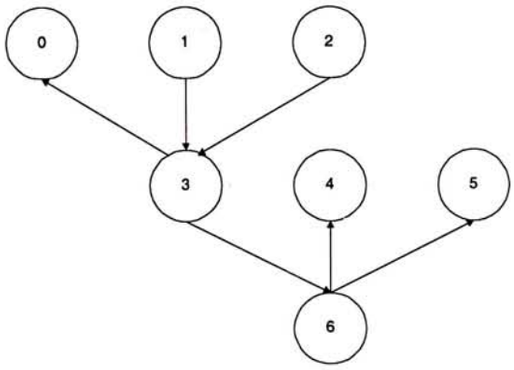
(a) The original structure



(b) GA



(c) Greedy0



(d) Greedy1

Figure 6.7: The original network structure of experiment 2 and the network structures found by the best run of different approaches

Variable 0: [0.05-0.26] [0.27-0.56] [0.58-1.01]
Variable 1: [0.04-0.35] [0.65-0.96]
Variable 2: [0.18-0.53] [0.54-0.92]
Variable 3: [0.16-0.3] [0.31-0.4] [0.41-0.46] [0.47-0.53] [0.54-0.66] [0.67-0.87]
Variable 4: [0.02-0.18] [0.42-0.82] [0.83-0.97]
Variable 5: [0-0.41] [0.42-1]
Variable 6: [0.11-0.36] [0.37-0.63] [0.64-0.76] [0.77-0.93] [0.94-1.13] [1.14-1.44]

(a) GA

Variable 0: [0.05-1.01]
Variable 1: [0.04-0.31] [0.32-0.96]
Variable 2: [0.18-0.46] [0.47-0.62] [0.63-0.92]
Variable 3: [0.16-0.4] [0.41-0.62] [0.63-0.87]
Variable 4: [0.02-0.51] [0.52-0.73] [0.75-0.97]
Variable 5: [0-1]
Variable 6: [0.11-0.45] [0.46-0.79] [0.8-1.07] [1.08-1.44]

(b) Greedy0

Variable 0: [0.05-0.36] [0.37-0.73] [0.75-1.01]
Variable 1: [0.04-0.34] [0.35-0.96]
Variable 2: [0.18-0.48] [0.49-0.67] [0.68-0.92]
Variable 3: [0.16-0.4] [0.41-0.51] [0.52-0.64] [0.65-0.87]
Variable 4: [0.02-0.18] [0.42-0.73] [0.75-0.97]
Variable 5: [0-0.36] [0.37-0.68] [0.69-1]
Variable 6: [0.11-0.43] [0.44-0.8] [0.81-1.05] [1.06-1.44]

(c) Greedy1

Figure 6.8: The discretization policies of the best results of experiment 2

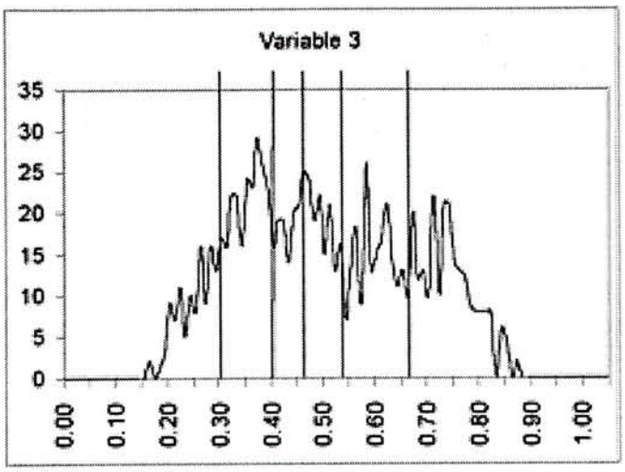


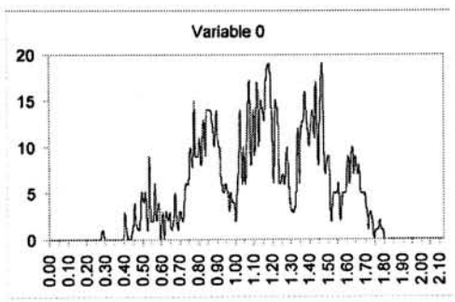
Figure 6.9: The ranges formed by the discretization policy of GA and the frequency distribution of variable 3

	mean	min.	max.	S.D.
GA	41363.1	41207	41653	150.40
Greedy0	43574.7	42584	44647	662.28
Greedy1	43077.2	42926	43436	144.70

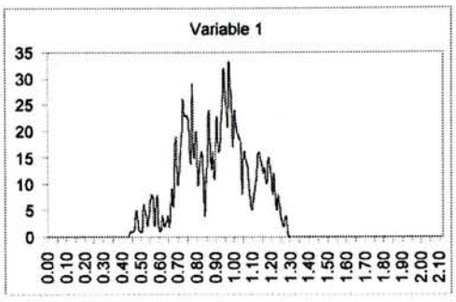
Table 6.3: Results of experiment 3

distributed in the range (0.1-0.6) and (0.4-1.0); Variable 7 is normally distributed in the range (0-0.2), (0.5-0.9) and (0.9-1.0). The actual frequency distributions of values in the data set are shown in Figure 6.10. The performance of ten trials of each approach is shown in Table 6.3. Again, GA can give the best results on the average score as well as the minimum score.

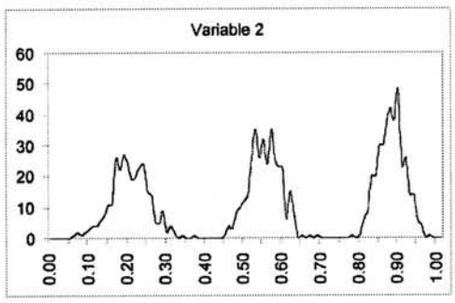
The best results give the network structures as shown in Figure 6.11. GA gives the structure that is the most similar to the original structure. The discretization policies of the best results are as shown in Figure 6.12. When comparing the discretization policies with the original distributions for generating the independent variables (variable 2,5,6,7), GA gives an extra range for variable 5 and 7, Greedy0 can reconstruct the ranges, while Greedy1 gives an extra range for variable 2 and 7.



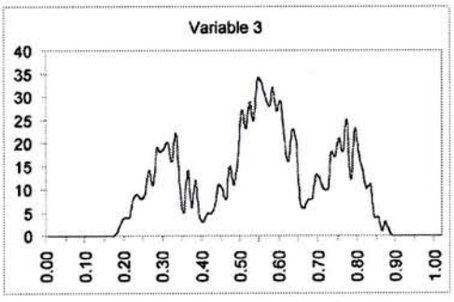
(a) Variable 0



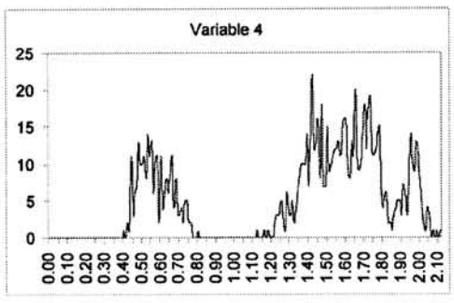
(b) Variable 1



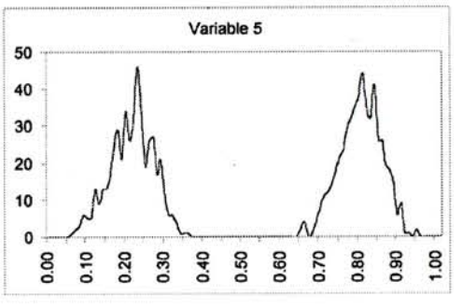
(c) Variable 2



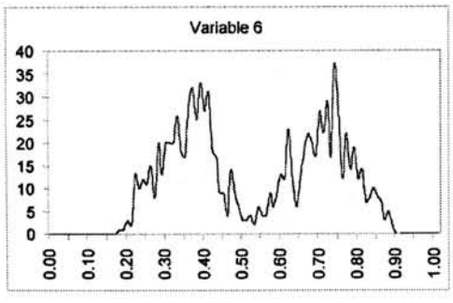
(d) Variable 3



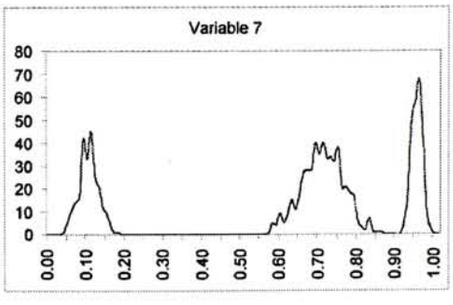
(e) Variable 4



(f) Variable 5

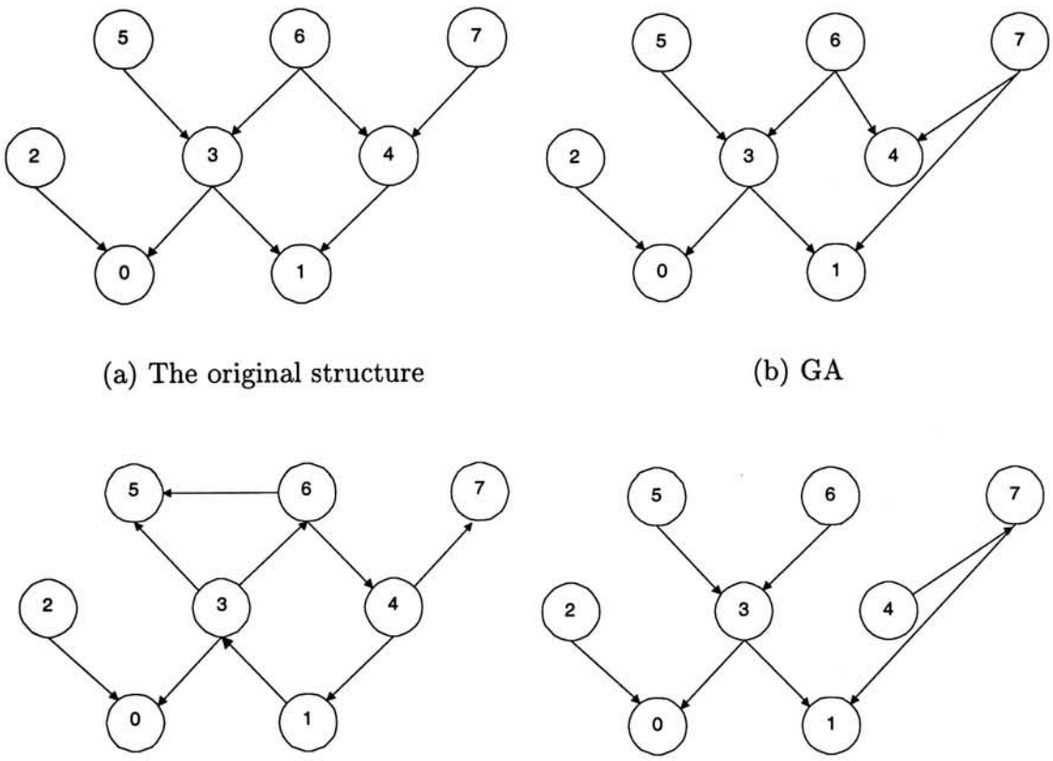


(g) Variable 6



(h) Variable 7

Figure 6.10: The frequency distribution of the variables of experiment 3



(a) The original structure

(b) GA

(c) Greedy0

(d) Greedy1

Figure 6.11: The original network structure of experiment 3 and the network structures found by the best run of different approaches

Variable 0: [0.28-0.7] [0.71-0.96] [0.97-1.12] [1.13-1.28] [1.29-1.53] [1.54-1.79]
Variable 1: [0.44-0.63] [0.64-0.83] [0.84-0.92] [0.93-0.99] [1-1.07] [1.08-1.28]
Variable 2: [0.06-0.31] [0.32-0.67] [0.69-0.98]
Variable 3: [0.18-0.41] [0.42-0.56] [0.57-0.67] [0.68-0.88]
Variable 4: [0.4-0.57] [0.58-1.12] [1.16-1.44] [1.45-1.64] [1.65-1.83] [1.84-2.1]
Variable 5: [0.06-0.36] [0.65-0.79] [0.8-0.95]
Variable 6: [0.18-0.52] [0.53-0.89]
Variable 7: [0.04-0.57] [0.58-0.71] [0.72-0.85] [0.86-0.99]

(a) GA

Variable 0: [0.28-0.7] [0.71-0.96] [0.97-1.22] [1.23-1.3] [1.31-1.56] [1.57-1.79]
Variable 1: [0.44-0.63] [0.64-0.84] [0.85-0.93] [0.94-1.05] [1.06-1.28]
Variable 2: [0.06-0.31] [0.32-0.78] [0.8-0.98]
Variable 3: [0.18-0.42] [0.43-0.66] [0.67-0.88]
Variable 4: [0.4-0.57] [0.58-0.8] [1.12-1.45] [1.46-1.68] [1.69-1.91] [1.92-2.1]
Variable 5: [0.06-0.35] [0.36-0.95]
Variable 6: [0.18-0.55] [0.56-0.89]
Variable 7: [0.04-0.18] [0.57-0.86] [0.92-0.99]

(b) Greedy0

Variable 0: [0.28-0.71] [0.72-0.95] [0.96-1.06] [1.07-1.23] [1.24-1.35] [1.36-1.56] [1.57-1.79]
Variable 1: [0.44-0.63] [0.64-0.84] [0.85-0.9] [0.91-1.06] [1.07-1.28]
Variable 2: [0.06-0.37] [0.45-0.52] [0.53-0.78] [0.8-0.98]
Variable 3: [0.18-0.44] [0.45-0.66] [0.67-0.88]
Variable 4: [0.4-1.16] [1.18-1.32] [1.33-1.59] [1.6-1.71] [1.72-1.86] [1.87-2.1]
Variable 5: [0.06-0.35] [0.36-0.95]
Variable 6: [0.18-0.58] [0.59-0.89]
Variable 7: [0.04-0.58] [0.59-0.66] [0.67-0.83] [0.84-0.99]

(c) Greedy1

Figure 6.12: The discretization policies of the best results of experiment 3

	Experiment 1	Experiment 2	Experiment 3
GA	5 minutes	70 minutes	90 minutes
Greedy0	15 seconds	15 minutes	30 minutes
Greedy1	15 seconds	15 minutes	30 minutes

Table 6.4: Execution time of the three approaches

6.3.4 Comparison between the GA approach and the greedy approach

From the results of these experiments, we can see that our new GA approach performs better than the greedy approach. When comparing the average score, GA is better than the two greedy approaches, except in experiment 1 where the difference is insignificant. When comparing the best trial in these experiments, GA can give the best result that the greedy approach cannot produce. In experiment 2 and 3, the data set is generated artificially under a network structure and special probability distributions. The network structures given by GA in experiment 2 and 3 are more similar to the original structures, and GA can give ranges similar to the underlying probability distributions in most of the independent variables. This shows that GA can successfully construct appropriate network structure and discretization policy from the data. Nevertheless, the original network structure and the underlying probability distributions may not give the best MDL score, and can only be references for comparisons.

The experiment results also confirm a deficiency of the greedy approach: the greedy approach depends greatly on the initial discretization. In these experiments, the standard derivation of Greedy0 is the largest. The greedy approach has more fluctuations than the GA approach when given a random initial discretization. From a poor discretization policy, the greedy approach does not have any technique to escape and thus gives a poor result, while the parallel search in GA approach can search for several local optima and gives a better result. When given a better initial discretization, such as in Greedy1, a better result is achieved. Since we cannot guarantee that we can start with a good initial discretization, the GA approach should be a better method to perform the optimization.

The approximation time for each execution in a Sun Ultra 1/140 is shown in Table 6.4. The execution time of the greedy approach is better than the GA approach. The execution time for both approaches is mainly spent on calculating the MDL score, as each evaluation needs to loop over every training case. In the GA approach, the number of fitness evaluations depends on the population size, the number of generations, and the number of iteration between network structure learning and discretization policy learning. In the greedy approach, the number of MDL score calculations depends on the number of values of each variable (because the greedy approach tests all the possible splits for each variable), the number of variables, and the number of iterations between network structure learning and discretization policy learning. Thus the execution time is a disadvantage of the GA approach.

Chapter 7

Medical Data Mining System

The approaches for rule learning and Bayesian network learning described in previous chapters have been combined into a knowledge discovery system. Figure 7.1 shows the steps in this system. Real-life data are collected in the first step. Then, the data must be preprocessed before analyses can be performed. The third and fourth steps induce knowledge from the preprocessed data. The Causality and Structure Analysis step learns the overall relationships between the variables. The GA approach described in Chapter 6 is employed to learn a Bayesian network from the nominal or continuous data. Based on this knowledge, the user can specify the target relationships he wants to know by formulating a grammar. The Rule

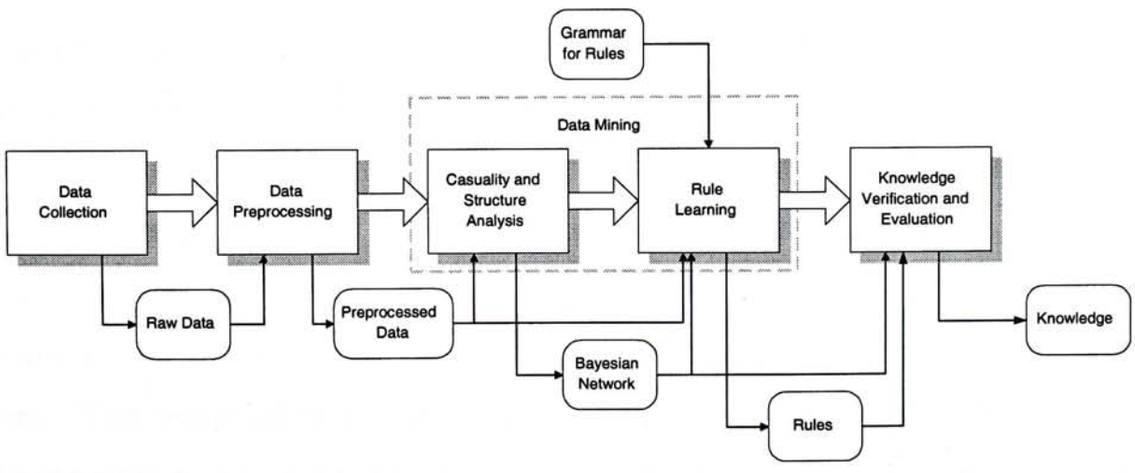


Figure 7.1: The knowledge discovery process

Learning step learns a set of significant rules from the data. The approach described in Chapter 5 is employed. The grammar can guide the format of the rules to be learned. In the fifth step, the learned knowledge is verified and evaluated by the domain experts. The domain experts may discover and correct mistakes in the learned knowledge. On the other hand, the learned knowledge can refine the existing domain knowledge. Finally, the constructed Bayesian network can be used to perform reasoning under uncertainty, and the induced rules can be incorporated into an expert system for decision making.

The use of grammar can ensure syntactical correctness in the rule, but not semantical correctness. It is desirable to eliminate meaningless rules in the search process. This requires a certain degree of knowledge on the causalities between the attributes. Causality and structure analysis in our data mining system can provide this knowledge. The Bayesian network may provide an overview of the relationships among the attributes. For example, if we know that attribute A is not related to any other attributes, then we don't need to learn rules about A. If we know attribute B should depend on attributes C and D, then we can specify a rule format like 'if <attribute C descriptor> and <attribute D descriptor>, then <attribute B descriptor>'.

The temporal order among attributes can also provide knowledge to increase the learning efficiency. For example, in a medical domain, the rule "if treatment is plaster, then diagnosis is radius fracture" is inappropriate. This rule does not make sense, because an operation is taken based on the treatment, not the other way round. In general, an event that occurs later will not be a cause of an event occurred earlier! Thus, we can order the attributes according to the temporal relationship. The grammar should be designed such that an attribute is not placed in the 'if' part if it occurs later than the attribute in the 'then' part. This temporal order can be represented easily in the grammar. Both of the causality model and temporal order may significantly reduce search space and prune meaningless rules.

The described data mining technology has been applied to real-life medical

Name	Type	Description	Possible Value
Sex	Nominal	Sex	'M' or 'F'
Age	Numeric	Age	Between 0 to 16 years old
Admday	Date	Admission date	Between year 1984 to 1996; Divided into four parts: Day, Month, Year and Weekday
Stay	Numeric	Length of staying in hospital	Between 0 to 1081 days
Diagnosis	Nominal	Diagnosis of fracture	10 different values, based on the location of fracture
Operation	Nominal	Operation	'CR' (Simple Closed Reduction), 'CR+K-wire' (Closed Reduction with K-wire), 'CR+POP' (Closed Reduction with POP), 'OR' (Open Reduction) or Null (no operation)
Surgeon	Nominal	Surgeon	One of 61 surgeons or Null if no operation
Side	Nominal	Side of fracture	'Left', 'Right', 'Both' or 'Missing'

Table 7.1: Attributes in the fracture database.

databases. The following two sections are two case study of knowledge discovery from a fracture database and a scoliosis database.

7.1 A Case Study on the Fracture Database

The fracture database consists of records of children with limb fractures, admitted to the Prince of Wales Hospital of Hong Kong in the period 1984-1996. This data can provide information for the analysis of children fracture patterns. This database has 6500 records and 8 attributes, which are listed in Table 7.1.

7.1.1 Results of Causality and Structure Analysis

The relationships among the attributes are analyzed by learning a Bayesian network. We have used a population size of 50 for both MDLEP and GA. The result cannot be improved after an execution of 10 hours. The discovered network structure is drawn in Figure 7.2. Day, Month, Weekday and Year refer to different parts of the admission date. The discretization policy is shown in Table 7.2. The age is divided into 0-4, 5-9, 10-12 and 13-16. The day and month are discretized

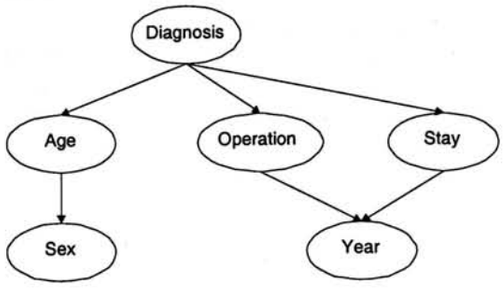


Figure 7.2: The best network structure for the fracture database

`Age`: [0-4] [5-9] [10-12] [13-16]
`Day`: [1-31]
`Month`: [1-12]
`Year`: [1984-1987][1988-1991][1992-1996]
`Stay`: [0-3] [4-12] [13-1081]

Table 7.2: Discretization policy of the fracture database

into just one range, which means that they are not involved in any relationship in the Bayesian network. Year is divided into 3 ranges. Stay is divided into 3 ranges.

From the network structure constructed, the following relationships are observed:

- `Diagnosis` implies `Operation` and `Stay`. Different fractures are treated with different operations, and require different time for recovery.
- `Diagnosis` can imply the value of `Age`. Some fractures are more frequently occurred in particular age groups.
- The value of `Age` can imply the value of `Sex`. It is observed that the young patients are more likely to be female, and elder patients are more likely to be male.
- `Operation` and `Stay` can determine `Year`. It is observed from the database that the length of stay in hospital is longer in the year 1985, 1986 and 1994, and open-reduction occurs more frequently for earlier years.

About	No. of Rules	*cf*			*cf/prob*			*support*		
		mean	max	min	mean	max	min	mean	max	min
Diagnosis	2	45.6%	51.4%	39.8%	1.6	1.7	1.4	9.2%	10.0%	8.4%
Operation	8	42.6%	74.0%	28.0%	2.0	2.9	1.1	5.4%	16.2%	3.2%
Stay	7	71.1%	81.1%	47.0%	2.5	7.0	1.4	4.5%	8.7%	3.1%

Table 7.3: Summary of the rules for the fracture database

7.1.2 Results of Rule Learning

Based on the learned Bayesian network, we observed a causality model between diagnosis, operation and stay. We wished to learn knowledge about these attributes. In addition, the temporal order gives extra knowledge on how the rules should be formulated. The attributes can be divided into three time stages: a diagnosis is first given to the patient, then an operation is performed, and after that the patient stays in the hospital. This knowledge leads to three causality models. Firstly, sex, age and admission date are the possible causes of diagnosis. Secondly, these three attributes and diagnosis are the possible causes of operation and surgeon. Thirdly, length of stay has all other attributes as the possible causes. A grammar (see Appendix B.1) is written as a template for these three kinds of rules. We have used a population size of 300 to run for 50 generations in the rule learning step. The execution time was about 3 hours on a Sun Ultra 1/140 for the 6500 records. The results are listed in Table 7.3.

Two interesting rules about diagnosis are found. The one with the highest confidence is:

```
If age is between 2 and 5, then diagnosis is Humerus.  (cf=51.43%)
```

The confidences of the rules about diagnosis are just around 40%-50%. It is partly because there are actually no strong rules affecting the value of diagnosis. However the ratio *cf/prob* shows that the patterns discovered deviated significantly from the average. We found that humerus fracture is the most common fracture for children between 2 and 5 years old. Radius fracture is the most common fracture for boys between 11 and 13.

Eight interesting rules about operation are found. The one with the highest

confidence is:

If age is between 0 and 7, and admission year is between 1988 and 1993, and diagnosis is Radius, then operation is CR+POP. (cf=74.05%)

These rules suggest that radius and ulna fractures are usually treated with CR+POP (i.e. plaster). Operation is usually not needed for tibia fracture. Open reductions are more common for elder children with age larger than 11, while young children with age lower than 7 have a higher chance of not needing operations. We did not find any interesting rules about surgeons, as the surgeons for operation are more or less randomly distributed in the database.

Seven interesting rules about length of stay are found. The one with the highest confidence is:

If admission year is between 1985 and 1996, and diagnosis is Femur, then stay is more than 8 days. (cf=81.11%)

The rules about the length of stay suggest that Femur and Tibia fractures are serious injuries and have to stay longer in hospital. If open reduction is used, the patient requires longer time to recover because the wound has been cut open for operation. If no operation is needed, it is likely that the patient can return home within one day. Relatively, radius fracture requires a shorter time for recovery.

The results have been evaluated by the medical experts. Previous analyses on fracture patterns only gave an overall injury pattern. Our system automatically uncovered relationships between different attribute values. The rules provide interesting patterns that were not recognized before. It clearly demonstrated the treatment pattern and rules of decision making. It can provide a good monitor of the change of pattern if the data mining process is continued longitudinally over the years. It also helps to provide the information for setting up a knowledge-based instruction system to help young doctors in training to learn the rules in diagnosis and treatment.

Name	Explanation and possible values
Sex	Sex 'M' or 'F'
Age	Age Positive integer
Lax	Joint Laxity Integer between 0 and 3
1stCurveT1	Whether 1st curve started at vertebra T1 Y or N
1stMCGreater	Whether the degree of 1st Major Curve > 2nd Major Curve Y or N
L4Tilt	Whether vertebra L4 is tilted Y or N
1stMCDeg	Degree of 1st Major Curve Positive integer
2ndtMCDeg	Degree of 2nd Major Curve Positive integer
1stMCApex	Apex of 1st Major Curve Any vertebra (vertebras are coded with T1-T12 or L1-L5)
2ndMCApex	Apex of 2nd Major Curve Null or any vertebra
Deg1	Degree of 1st Curve Positive integer
Deg2	Degree of 2nd Curve Positive integer
Deg3	Degree of 3rd Curve Positive integer
Deg4	Degree of 4th Curve Positive integer
Class	Scoliosis Classification K-I, K-II, K-III, K-V, TL, L
Mens	Period of Menstruation Positive integer; -9 for no menstruation yet; 99 for male
TSI	Trunk Shift (measures the displacement of the curve) Positive integer
TSIDir	Trunk Shift Direction Null, left or right
RI	Risser Sign (measures the maturity of the patient) Integer between 0 and 5
Treatment	Treatment Observation, surgery or bracing

Table 7.4: Attributes in the Scoliosis database

7.2 A Case Study on the Scoliosis Database

The data mining process has also been applied to the database of Scoliosis patients. Scoliosis refers to the spinal deformation. A Scoliosis patient has one or several curves in his spine. Among them, the curves with severe deformations are identified as major curves. The database stores measurements on the patients, such as the number of curves, the curve locations, degrees and directions. It also records the maturity of the patient, the class of Scoliosis and the treatment. The database has about 500 records. According to the domain expert, 20 attributes are useful and extracted from the database in the preprocessing step. They are shown in Table 7.4.

7.2.1 Results of Causality and Structure Analysis

In this database, the attributes `Age`, `1stMCDeg`, `2ndMCDeg`, `Deg1` to `Deg4` and `Mens` are continuous variables. For the attributes measuring degrees, the value 0 is a special value as it means the curve does not exist. For `Mens`, the values -9 and 99 have special meanings, which indicate no menstruation. These values are specially handled by always placing a 1 in the corresponding positions of the bit string in GA, such that they are always discretized from other values. Then each continuous variable is initially discretized into 3 ranges.

The learning of network structure and discretization policy are alternated for 20 iterations. For the learning of network structure using MDLEP, we have used a population of 50 to run for 100 generations. In each iteration of the learning of discretization policy using GA, the population size is 50 and the number of generation is 10. The number of generations is small, but the learning is iterated 20 times, thus there should be enough generations for convergence. The best Bayesian network structure learned from this data set is shown in Figure 7.3. The discretization policy is shown in Table 7.5. The age is divided into 0-12 (child), 13-16 (adolescence), 17-21 and over 22. The degrees and `Mens` are divided into different ranges.

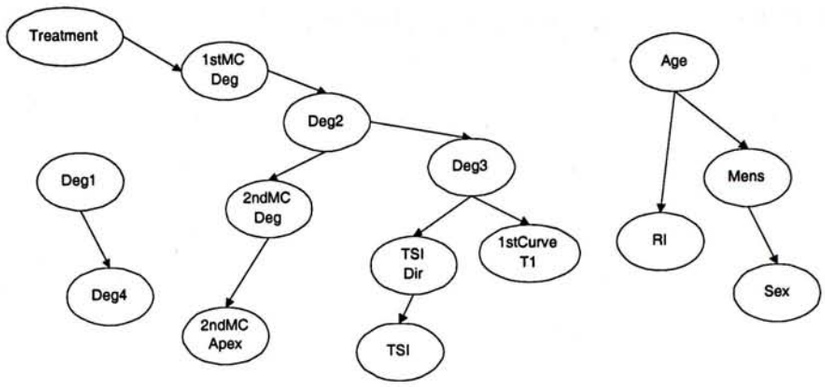


Figure 7.3: The best network structure for the Scoliosis database

`Age`:	[0-12] [13-16] [17-21] [22-41]
`1stMCDeg`:	[5-13] [14-29] [30-35] [36-52] [53-112]
`2ndMCDeg`:	[0-0] [5-23] [24-36] [37-65]
`Deg1`:	[3-11] [12-35] [36-52] [54-112]
`Deg2`:	[0-0] [2-26] [27-36] [37-52] [53-93]
`Deg3`:	[0–0][3-21] [22-60]
`Deg4`:	[0–0][13-34]
`Mens`:	[-9 - -9] [0-4] [5-30] [99-99]

Table 7.5: Discretization policy of the Scoliosis database

From the network structure constructed, the following relationships are observed:

- `Age` can determine `Mens` and `RI` (the maturity), and the value of `Mens` can imply `Sex`.
- The value of `Deg1` can imply the value of `Deg4`. In the database only a few records have values of `Deg4` larger than 0. All of these records have large values on `Deg1`.
- `Operation` can determine the value of `1stMCDeg`. If `Operation` equals to observation, the value `1stMCDeg` is smaller. If `Operation` equals to surgery, the value of `1stMCDeg` is large.
- The value of `1stMCDeg` affects the value of `Deg2`. It is observed that if the value of the first major curve is small, the degree of the second curve must be small. `Deg2` should not be larger than `1stMCDeg`. Otherwise the first

major curve should be the second curve, and `1stMCDeg` should be equal to `Deg2`.

- `Deg2` implies the value of `2ndMCDeg`, since most of the time the second major curve is the second curve. The value of `2ndMCDeg` also closely related with `2ndMCApex` (the location of second major curve). If `2ndMCDeg` equals to 0, the patient does not have the second major curve, and thus `2ndMCApex` must be null.

- `Deg2` can imply the value of `Deg3`, since if `Deg2` is small, most likely `Deg3` is zero.

- `Deg3` can imply the value of `1stCurveT1`. If `Deg3` is large, the spine has three or more curve, and most likely the first curve starts at the first vertebra T1.

- `Deg3` can imply the value of `TSIDir`. If `Deg3` is small, most of the time the direction of trunk shift is null

- `TSIDir` can imply `TSI` because if direction of trunk shift is null, `TSI` should be 0.

- `Treatment` can imply `1stMCDeg`. If treatment is bracing, most likely the degree of the first major curve is small. In contrast, if operation is needed, the degree of the first major curve is usually large.

7.2.2 Results of Rule Learning

The medical experts are interested to discover knowledge about classification of Scoliosis and treatment. Scoliosis can be classified as Kings, Thoracolumbar(TL) and Lumbar(L), while Kings can be further subdivided into K-I, II, III, IV and V. Treatment can be observation, surgery and bracing. The determinations of these two attributes are complicated. Unfortunately, the Bayesian network does not discover any significant relationship for these two variables. According to the domain expert, classification should be related to the attributes `1stCurveT1`,

Class	No. of Rules	*cf*			*support*			*prob*
		mean	max	min	mean	max	min	
King-I	5	94.84%	100%	90.48%	5.67%	10.73%	0.86%	28.33%
King-II	5	80.93%	100%	52.17%	6.61%	14.38%	1.07%	35.41%
King-III	4	23.58%	25.87%	16.90%	1.56%	2.58%	0.86%	7.94%
King-IV	2	24.38%	29.41%	19.35%	1.18%	1.29%	1.07%	2.79%
King-V	5	54.13%	62.50%	45.45%	0.97%	1.07%	0.86%	6.44%
TL	1	41.18%	41.18%	41.18%	1.50%	1.50%	1.50%	2.15%
L	3	54.04%	62.50%	45.45%	2.00%	2.79%	1.07%	4.51%

Table 7.6: Results of the rules for Scoliosis classification

`1stMCGreater`, `L4Tilt`, `1stMCDeg`, `2ndMCDeg`, `1stMCApex` and `2ndMCApex`, and treatment should be related to age, laxity, degrees of the curves, maturity of the patient, displacement of the vertebra and the class of Scoliosis. This domain knowledge can be easily incorporated in the design of the rule grammar. There are two types of rules, one for classification of Scoliosis and the other for suggesting treatment. The grammar is outlined in Appendix B.2.

The population size used in the rule learning step is 100 and the maximum number of generations is 50. The execution time was about one hour on a Sun Ultra 1/140. The results of rule learning from this database are listed below.

Rules for Scoliosis classification.

For each class of Scoliosis, a number of rules are mined. The results are summarized in Table 7.6. The rules are listed in Appendix A.4.1. An typical rule of this kind is:

```
if 1stMCGreater = N and 1stMCApex = T1-T8 and 2ndMCApex = L3-L4,
then King-I. (cf=100%)
```

For King-I and II, the rules have high confidence and generally match with the knowledge of medical experts. However the fourth rules of King-II is an unexpected rule for the classification of King-II. Under the conditions specified in the antecedents, our system found a rule with a confidence factor of 52% that the classification is King-II. However, the domain expert suggests the class should be King-V! After an analysis on the database, we revealed that serious data errors

existed in the current database and that some records contained an incorrect Scoliosis classification.

For King-III and IV, the confidence of the rules discovered is just around 20%. According to the domain expert, one common characteristic for these two classes is that there is only one major curve or the second major curve is insignificant. However there is no rigid definition for a 'major curve' and the concept of 'insignificant' is fuzzy. These depend on the interpretation of doctors. Because of the lack of this important information, the system cannot find accurate rules for these two classes. Another problem is that only a small number of patients in the database were classified to King-III or IV (see the values of *prob* in Table 7.6). The database cannot provide a large number of cases for training. Similar problems also existed for King-V, TL and L.

For the class King-V, TL and L, the system found rules with confidence around 40% to 60%. Nevertheless, the rules for TL and L show something different in comparison with the rules suggested by the clinicians. According to our rules, the classification always depends on the location of the *first major curve*, while according to the domain expert, the classification always depends on the *larger major curve.* After discussion with the domain expert, it is agreed that the existing rules are not defined clearly enough, and our rules are more accurate than them. Our rules provide hints to the clinicians to re-formulate their concepts.

Rules about treatment

The results of rules about treatment are summarized in Table 7.7. The rules are listed in Appendix A.4.2. An typical rule of this kind is:

```
If age=2-12 and Deg1=20-26 and Deg2=24-47 and Deg3=27-52 and Deg4=0,
then Bracing.  (cf=100%)
```

The rules for observation and bracing have very high confidence factors. However, the support is not high, showing that the rules only cover fragments of the cases. Our setting in our learning prefers accurate rules to general rules. If the user prefers more general rules, the weights in the fitness function can be tuned.

Type	No. of Rules	*cf*			*support*			*prob*
		mean	max	min	mean	max	min	
Observation	4	98.89%	100%	95.55%	3.49%	6.01%	1.07%	62.45%
Bracing	5	79.57%	100%	71.43%	1.03%	1.29%	0.86%	24.46%
Surgery	0	-	-	-	-	-	-	3.65%

Table 7.7: Results of the rules about treatment

For surgery, no interesting rule was found because only 3.65% of the patients are treated with surgery.

The biggest impact on the clinicians from the data mining analysis of the Scoliosis database is the fact that many rules set out in the clinical practice are not clearly defined. The usual clinical interpretation depends on the subjective experience. Data mining revealed quite a number of mismatches in the classification on the type of Kings curves. After a careful review by the senior surgeon it appears that the database entries by junior surgeons may not be accurate and that the data mining rules discovered are in fact more accurate! The classification rules must therefore be quantified. The rules discovered can therefore help in the training of younger doctors and act as an intelligent means to validate and evaluate the accuracy of the clinical database.

Chapter 8

Conclusion and Future Work

In this thesis, we have presented two approaches for learning rules and Bayesian networks from data. They both employ Evolutionary Computation as the search algorithms. A data mining system that can learn rules and Bayesian networks from data has been developed. Causality and Structure Analysis in the system learns a Bayesian network from the data. It focuses on the general causality model between the *variables*. In contrast, the rule learning step learns a set of rules from the data. It captures the specific behavior between particular *values* of the variables.

We have used Generic Genetic Programming (GGP) as the search algorithm for rule learning. The grammar used in GGP can provide a powerful knowledge representation. It can specify the format of the rules to be discovered. The format can be changed according to different domains, and the flexible grammar allows the representation of general concepts. Moreover, knowledge from domain experts can be very useful to data mining. The use of grammar allows the domain knowledge to be easily and effectively utilized. Furthermore, the user can specify the desirable rule format by composing a suitable grammar. This can increase the understandability and the usefulness of the discovered rules.

In many real-life situations, the available rules are general guidelines with many exceptional cases. The fitness function in the rule learning approach has been designed to learn such kind of knowledge. It compares the confidence of the

rule with the average probability, so as to search for the patterns deviated significantly from the normal. Since one rule is insufficient to represent the complete knowledge, token competition has been used to learn as many rules as possible. This technique can effectively and efficiently formulate niches in the population, such that different rules are evolved in the same population. This rule learning approach can successfully construct rules from data. The rules can represent the regularities in the database and provide interesting knowledge to the users.

The knowledge hidden in real-life database usually cannot be described completely by just a few rules. Building a complete model for such a database is difficult and usually results in a complicated model. Bayesian network is a knowledge representation that can be a complement to rules. Instead of capturing the interesting patterns between particular values of attributes, a Bayesian network gives a general view on the causality between attributes in a graphical model. It is easy to understand while it has a well-developed mathematical model. The Bayesian network representation requires the attributes to be discrete. We have extended the work on the Minimum Description Length (MDL) for discretizing continuous variables. We have investigated the use of Genetic Algorithm to optimize the MDL score for discretization. The experimental results show that Genetic Algorithm performs better than the greedy approach.

The rule learning approach and the Bayesian network learning approach have been combined in a data mining system. The Bayesian network learned from the causality and structure analysis can help the user to understand more on the relationships between attributes, and provide knowledge for guiding the search of rules. The causality presented in the Bayesian network, as well as the domain knowledge and the temporal relationships between attributes, can provide knowledge to the user to compose a suitable grammar for rule learning. A suitable grammar can prune the search space on meaningless rules and increase the search efficiency.

The data mining system has been applied to two real-life medical databases.

The results can provide interesting knowledge as well as suggestion for refinements to the existing knowledge. We also have found unexpected results that led to discovery of mistakes in the database. In the fracture database, the system automatically uncovered knowledge about the age effect on fracture, the relationship between diagnoses and operations, and the effect of diagnoses and operations on lengths of staying in the hospital. In the Scoliosis database, we have discovered new knowledge about the classification of Scoliosis and about the treatment. The discovered knowledge leads to refinements of the existing knowledge.

The approach for data mining can be improved in various aspects. The rule learning approach is based on GGP with a context free grammar. This grammar still may not be powerful enough to represent the hidden knowledge. The knowledge representation can be strengthened if context sensitive instead of context free grammar is implemented. The fitness function used in rule learning is far from perfect. A more solid fitness function should be defined by doing a more complete theoretical analysis. For the Bayesian network learning, the search is alternated between structure learning and discretization policy learning. The network structure is learned from a sub-optimal discretization policy, and vice versa. A better result can be obtained if we can designed a method to optimize both the network structure and discretization policy learning in a single step, although the search space is greatly increased in this way.

The usability of the data mining system can also be improved. The grammar in rule learning provides a powerful knowledge representation, but the users has to compose the grammar themselves to fit the problems. The construction of grammar can be simplified if a generic graphical user interface is provided. The time complexity is a major disadvantage of evolutionary algorithms. The execution speed can be improved if results of previous generations can be cached. Better methods for calculations of fitness should be designed such that the calculation can fully utilize the results of previous generations.

Bibliography

R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases, Santiage, Chile*, pages 487–499, September,1994.

R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 International Conference on Management of Data (SIGMOD 93)*, pages 207–216, 1993.

J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 1985.

L. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

R. R. Bouckaert. Properties of belief networks learning algorithms. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 102–109, 1994.

J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning - An Artificial Intelligence Approach*, chapter 1. Los Altos,Calif., 1983.

D. J. Cavicchio. *Adaptive search using simulated evolution.* PhD thesis, University of Michigan, Ann Arbor, 1970.

E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

M.S. Chen, J. Han, and S. Yu. Data mining : An overview from database perspective. *IEEE transactions on Knowledge and Data Engineering*, 8(6):866, December 1996.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

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

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

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

K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems, Dissertation Abstracts International 36(10), 5140B (University Microfilms No. 76-9381).* PhD thesis, University of Michigan, Ann Arbor, 1975.

J. F. Elder IV and D. Pregibon. A statistical perspective on KDD. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

U. M. Fayyad, G. Piatesky-Shapiro, and P. Smyth. From data mining to knowledge discovery : An overview. *AI Magazine*, pages 37–54, Fall 1996.

L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution.* New York: John Wiley and Sons, 1966.

D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Network*, 5:3–14, 1994.

W. J. Frawley, G. Piatesky-Shapiro, and C. J. Matheus. Knowledge discovery in databases : An overview. *AI Magazine*, pages 57–70, Fall 1992.

N. Friedman and M. Goldszmidt. Discretizing continuous attributes while learning Bayesian networks. In *International Conference on Machine Learning*, pages 157–165, 1996.

A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation*, 3:375–416, 1995.

D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the second International Conference on Genetic Algorithms*, pages 41–49, 1987.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

J. Han and Y. Fu. Discovery of multiple level association rules from large databases. In *Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland*, September,1995.

D. Heckerman and M. P. Wellman. Bayesian networks. *Communications of the ACM*, 38(3):27–30, March 1995.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

D. Heckerman. Bayesian networks for knowledge discovery. In *Advances in Knowledge Discovery and Data Mining*, pages 273–306. AAAI/MIT Press, 1996.

D. Heckerman. Bayesian networks for data mining. *Data Mining and Knowledge Discover*, 1:79–119, 1997.

E. Herskovits and G. Cooper. KUTATO: An entropy-driven system for construction of probabilistic expert systems from databases. Technical Report KSL-90-22, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, 1990.

J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*. Academic Press, 1978.

J. H. Holland. *Adaptation in Natural and Artificial Systems.* Bradford/MIT Press, 1992.

R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:91–104, 1993.

P. Hoschka and W. Klösgen. A support system for interpreting statistical data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

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

W. Klösgen. *Explora, A support system for Discovery in Databases, Version 1.1 User Manual.* GMD, Sankt Augustin, 1993. URL:

ftp://ftp.gmd.de/gmd/explora/EXPLORA-MANUAL.ps.gz.

J. R. Koza. *Genetic Programming : on the programming of computers by means of natural selection.* Bradford/MIT Press, 1992.

J. R. Koza. *Genetic Programming II : automatic discovery of reusable programs.* Bradford/MIT Press, 1994.

W. Lam and F. Bacchus. Learning Bayesian belief networks - an approach based on the MDL principle. *Computational Intelligence*, 10(3):269–293, 1994.

W. Lam, M. L. Wong, K. S. Leung, and P. S. Ngan. Discovering probabilistic knowledge from databases using evolutionary computation and minimum description length principle. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 786–794, 1998.

W. Lam. Bayesian network refinement via machine learning approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):240–251, 1998.

P. Larranaga, C. Kuijpers, R. Murga, and Y. Yurramendi. Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on System, Man, and Cybernetics - Part A: Systems and Humans*, 26(4):487–493, 1996a.

P. Larranaga, M. Poza, Y. Yurramendi, R. Murga, and C. Kuijpers. Structure learning of Bayesian network by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):9, 1996b.

K. S. Leung, Y. Leung, L. So, and K. F. Yam. Rule learning in expert systems using genetic algorithm: 1, concepts. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks(Iizuka, Japan)*, pages 201–204, 1992.

S. W. Mahfoud. Crowding and preselection revisited. *Parallel Problem Solving from Nature*, 2:27–36, 1992.

H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI Workshop on Knowledge Discovery in*

Databases, Seattle, Washington, Seattle, Washington, July 1994.

C.J. Merz and P.M. Murphy. *UCI Repository of machine learning databases.* University of California, Irvine, Dept. of Information and Computer Sciences, 1998. URL: http://www.ics.uci.edu/~mlearn/MLRepository.html.

R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 1041–1045, 1986.

R. S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing*, pages 125–128, 1969.

R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning - An Artificial Intelligence Approach*, chapter 4. Los Altos,Calif., 1983.

P. S. Ngan, W. Lam, M. L. Wong, K. S. Leung, and J. C. Y. Cheng. Medical data mining using evolutionary computation. *Artificial Intelligence in Medicine, special issue of data mining in medicine, To appear*, 1998a.

P. S. Ngan, M. L. Wong, K. S. Leung, and J. C. Y. Cheng. Using grammar based genetic programming for data mining of medical knowledge. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 254–259, 1998b.

J. S. Park, M. S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *Proceedings of the ACM-SIGMOD Conference on Management of Data, San Jose, California*, May,1995.

G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

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

J. R. Quinlan. *C4.5: programs for machine learning.* San Mateo, Calif. : Morgan Kaufmann Publishers, 1993.

G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In *Uncertainty in Artificial Intelligence 3*, pages 175–182. North-Holland, Amsterdam, 1989.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (In English: Evolution Strategy: Optimization of technical systems by means of biological evolution).* Stuttgart: Fromman-Holzboog, 1973.

J. Rissanen. Modeling by shortest data description. *Automatica*, pages 465–471, 1978.

H. P. Schwefel. *Numerical Optimization of Computer Models.* Chichester: Wiley, 1981.

M. Singh and M. Valtorta. An algorithm for the construction of Bayesian network structures from data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 259–265, 1993.

S. F. Smith. *A Learning System based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

S. F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth International Conference On Artificial Intelligence.* Morgan Kaufmann, 1983.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search.* Springer-Verlag, 1993.

R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1996.

W. A. Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 303–309, 1993.

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 CMU-CS-91-197, Carnegie Mellon University, 1991.

M. L. Wong and K. S. Leung. Inducing logic programs with genetic algorithms: The genetic logic programming system. *IEEE Expert*, 10(5):68–76, 1995.

M. L. Wong and K. S. Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5:143–180, 1997.

M. L. Wong, W. Lam, and K. S. Leung. Using evolutionary computation and minimum description length principle for data mining of probabilistic knowledge. *submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

M. L. Wong. *Evolutionary program induction directed by logic grammars.* PhD thesis, The Chinese University of Hong Kong, 1995.

Q. Wu, P. Suetens, and A. Oosterlinck. Integration of heuristic and bayesian approaches in a pattern-classification system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

J. M. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

Appendix A

The Rule Sets Discovered

A.1 The Best Rule Set Learned from the Iris Database

1. if petal width is between 0.08 and 0.77, then class is Iris-setosa.

   Fitness: 1.50

   Confidence: 100%; Support: 30%; Probability of consequent: 30%

2. if petal length is between 1.98 and 4.97, and petal width is between 0.18 and 1.66, then class is Iris-vericolor.

   Fitness: 1.37

   Confidence: 100%; Support: 35%; Probability of consequent: 35%

3. if sepal width is between 2.33 and 3.16, then class is Iris-virginica.

   Fitness: 0.43

   Confidence: 49.06%; Support: 26%; Probability of consequent: 35%

4. if any, then class is Iris-virginica.

   Fitness: 0.35

   Confidence: 35%; Support: 35%; Probability of consequent: 35%

A.2 The Best Rule Set Learned from the Monk Database

A.2.1 Monk1

1. if jacket_color = 1, then positive.

Fitness: 11.33

Confidence: 100%; Support: 23.39%; Probability of consequent: 50%

2. if head_shape = 1 and body_shape = 1, then positive.

   Fitness: 9.93

   Confidence: 100%; Support: 7.26%; Probability of consequent: 50%

3. if head_shape = 2 and body_shape = 2, then positive.

   Fitness: 8.98

   Confidence: 100%; Support: 12.10%; Probability of consequent: 50%

4. if head_shape = 3 and body_shape = 3, then positive.

   Fitness: 8.59

   Confidence: 100%; Support: 13.70%; Probability of consequent: 50%

5. if any, then negative.

   Fitness: 0.51

   Confidence: 50%; Support: 50%; Probability of consequent: 50%

A.2.2 Monk2

1. if head_shape $\neq$ body_shape and is_smiling = 1 and holding $\neq$ 1 and jacket_color = 2 and has_tie $\neq$ 1, then positive.

   Fitness: 15.59

   Confidence: 100%; Support: 4.73%; Probability of consequent: 37.87%

2. if head_shape = 2 and body_shape $\neq$ 1 and is_smiling $\neq$ 2 and holding $\neq$ 1 and jacket_color $\neq$ 1 and has_tie $\neq$ 2, then positive.

   Fitness: 15.58

   Confidence: 100%; Support: 3.55%; Probability of consequent: 37.87%

3. if head_shape $\neq$ body_shape and is_smiling $\neq$ 1 and jacket_color = 1 and has_tie $\neq$ 1, then positive.

   Fitness: 15.58

   Confidence: 100%; Support: 2.96%; Probability of consequent: 37.87%

4. if body_shape $\neq$ 1 and is_smiling $\neq$ 1 and holding = 2 and jacket_color = 1 and has_tie $\neq$ 2, then positive.

   Fitness: 15.57

   Confidence: 100%; Support: 2.37%; Probability of consequent: 37.87%

5. if head_shape = 1 and is_smiling ≠ 2 and holding ≠ 1 and jacket_color = 3 and has_tie ≠ 1, then positive.

   Fitness: 15.56

   Confidence: 100%; Support: 1.78%; Probability of consequent: 37.87%

6. if body_shape = 1 and is_smiling = 1 and jacket_color = 3 and has_tie = 2, then positive.

   Fitness: 15.56

   Confidence: 100%; Support: 1.78%; Probability of consequent: 37.87%

7. if head_shape ≠ 1 and body_shape ≠ 1 and is_smiling ≠ 1 and holding = 3 and jacket_color = 1, then positive.

   Fitness: 15.56

   Confidence: 100%; Support: 1.78%; Probability of consequent: 37.87%

8. if head_shape = 1 and is_smiling ≠ 2 and holding ≠ 1 and jacket_color = 4 and has_tie ≠ 1, then positive.

   Fitness: 15.56

   Confidence: 100%; Support: 1.18%; Probability of consequent: 37.87%

9. if head_shape = 3 and body_shape ≠ 3 and is_smiling ≠ 2 and jacket_color ≠ 1 and has_tie = 2, then positive.

   Fitness: 5.05

   Confidence: 87.50%; Support: 4.14%; Probability of consequent: 37.87%

10. if head_shape ≠ body_shape and holding ≠ 1 and jacket_color = 2 and has_tie = 1, then positive.

    Fitness: 3.96

    Confidence: 70%; Support: 4.14%; Probability of consequent: 37.87%

11. if body_shape ≠ 1 and is_smiling ≠ 1 and holding = 2 and jacket_color ≠ 2 and has_tie ≠ 2, then positive.

    Fitness: 2.75

    Confidence: 75%; Support: 3.55%; Probability of consequent: 37.87%

12. if head_shape ≠ body_shape and is_smiling = 1 and holding ≠ 1 and jacket_color = 2, then positive.

    Fitness: 2.37

    Confidence: 91.67%; Support: 6.50%; Probability of consequent: 37.87%

13. if head_shape $\neq$ body_shape and holding $\neq 2$ and jacket_color $= 2$ and has_tie $= 1$, then positive.

    Fitness: 1.35

    Confidence: 83.33%; Support: 2.96%; Probability of consequent: 37.87%

14. if body_shape $= 1$ and is_smiling $\neq 1$ and jacket_color $\neq 1$ and has_tie $= 2$, then positive.

    Fitness: 1.13

    Confidence: 50%; Support: 3.55%; Probability of consequent: 37.87%

15. if any, then negative.

    Fitness: 0.63

    Confidence: 62.13%; Support: 62.13%; Probability of consequent: 62.13%

A.2.3 Monk3

1. if body_shape $\neq 3$ and is_smiling $= 2$ and jacket_color $\neq 4$, then positive.

   Fitness: 11.46

   Confidence: 100%; Support: 22.30%; Probability of consequent: 49.59%

2. if head_shape $\neq$ body_shape and holding $= 1$ and jacket_color $= 3$, then positive.

   Fitness: 6.76

   Confidence: 100%; Support: 4.13%; Probability of consequent: 49.59%

3. if body_shape $\neq 3$ and holding $\neq 2$ and jacket_color $= 2$, then positive.

   Fitness: 6.06

   Confidence: 100%; Support: 12.40%; Probability of consequent: 49.59%

4. if head_shape $\neq 1$ and holding $= 1$ and jacket_color $= 3$, then positive.

   Fitness: 4.51

   Confidence: 100%; Support: 4.13%; Probability of consequent: 49.59%

5. if body_shape $\neq 3$ and jacket_color $\neq 4$, then positive.

   Fitness: 2.68

   Confidence: 91.94%; Support: 47.10%; Probability of consequent: 49.59%

6. if body_shape $\neq 3$ and jacket_color $= 2$ and has_tie $\neq 1$, then positive.

   Fitness: 1.62

   Confidence: 100%; Support: 11.57%; Probability of consequent: 49.59%

7. if head_shape $\neq$ 2 and body_shape $\neq$ 3 and holding $\neq$ 3 and jacket_color = 2, then positive.

   Fitness: 0.87

   Confidence: 100%; Support: 10.74%; Probability of consequent: 49.59%

8. if any, then negative.

   Fitness: 0.51

   Confidence: 50.41%; Support: 50.40%; Probability of consequent: 50.40%

A.3 The Best Rule Set Learned from the Fracture Database

A.3.1 Type I Rules: About Diagnosis

1. Humerus

   if age is between 2 and 5, then diagnosis is Humerus .

   Fitness: 3.48

   Confidence: 39.75%; Support: 8.42%; Probability of consequent: 23.43%

2. Radius

   if sex is M, and age is between 11 and 13, then diagnosis is Radius .

   Fitness: 3.04

   Confidence: 51.43%; Support: 10.01%; Probability of consequent: 36.10%

A.3.2 Type II Rules : About Operation/Surgeon

1. Radius vs. CR+POP

   if age is between 0 and 7, and admission year between 1988 and 1993, and diagnosis is Radius, then operation is CR+POP.

   Fitness: 8.56

   Confidence: 50.61%; Support: 3.19%; Probability of consequent: 17.72%

2. Tibia vs. No Operation

   if age is between 1 and 7, and diagnosis is Tibia, then operation is Null (i.e. no operation).

   Fitness: 7.86

   Confidence: 74.05%; Support: 3.78%; Probability of consequent: 38.11%

3. Ulna vs. CR+POP

   if age is between 1 and 12, and admission year between 1989 and 1992, and diagnosis is Ulna, then operation is CR+POP.

   Fitness: 7.19

   Confidence: 47.37%; Support: 3.50%; Probability of consequent: 17.72%

   if diagnosis is Ulna, then operation is CR+POP.

   Fitness: 4.23

   Confidence: 36.17%; Support: 7.40%; Probability of consequent: 17.72%

4. Radius vs. CR+K-Wire

   if admission year is between 1992 and 1994, and diagnosis is Radius, then operation is CR+K-Wire.

   Fitness: 4.10

   Confidence: 34.03%; Support: 3.83%; Probability of consequent: 16.23%

5. Humerus vs. CR+K-Wire

   if diagnosis is Humerus, then operation is CR+K-Wire.

   Fitness: 2.52

   Confidence: 27.96%; Support: 6.06%; Probability of consequent: 16.23%

6. Ulna vs. OR

   if age is between 11 and 15, and diagnosis is Ulna, then operation is OR.

   Fitness: 3.24

   Confidence: 33.20%; Support: 3.25%; Probability of consequent: 18.26%

7. Age vs. OR

   if sex is M, and age is between 13 and 17, and admission year between 1985 and 1989, then operation is OR.

   Fitness: 2.57

   Confidence: 30.53%; Support: 3.22%; Probability of consequent: 18.26%

8. Age vs. No Operation

   if age is between 0 and 7, then operation is Null (i.e. no operation).

   Fitness: 1.08

   Confidence: 43.33%; Support: 16.22%; Probability of consequent: 38.11%

A.3.3 Type III Rules : About Stay

1. Femur vs. Stay

   if admission year between 1985 and 1996, and diagnosis is Femur , then stay is between 8 and 2000 days. (i.e. stay 8 days or more, since 2000 is the maximum value of stay)

   Fitness: 21.99

   Confidence: 70.87%; Support: 3.14%; Probability of consequent: 10.24%

   if diagnosis is Femur , then stay is between 5 and 2000 days. (i.e. stay 5 days or more)

   Fitness: 18.70

   Confidence: 80.99%; Support: 3.30%; Probability of consequent: 19.22%

2. Tibia vs. Stay

   if age between 5 and 12, and diagnosis is Tibia, then stay is between 3 and 2000. (i.e. stay 3 days or more)

   Fitness: 8.93

   Confidence: 78.92%; Support: 5.05%; Probability of consequent: 39.15%

3. OR vs. Stay

   if age between 2 and 14, and diagnosis is Humerus, and operation is OR, then stay is between 3 and 25 days.

   Fitness: 8.86

   Confidence: 75.57%; Support: 3.52%; Probability of consequent: 36.51%

   if admission is between 1985 and 1987, and operation is OR, then stay is between 3 and 10 days.

   Fitness: 6.99

   Confidence: 65.52%; Support: 3.47%; Probability of consequent: 33.85%

   if operation is OR, then stay is between 3 and 25 days.

   Fitness: 6.13

   Confidence: 64.90%; Support: 12.22%; Probability of consequent: 36.51%

4. No operation vs. Stay

   if age is between 10 and 14, and admission year is between 1987 and 1996, and diagnosis is Radius, and operation is Null, then stay is between 0 and 1 day.

   Fitness: 9.55

   Confidence: 77.00%; Support: 3.09%; Probability of consequent: 35.65%

   if operation is Null, then stay is between 0 and 1 day.

Fitness: 3.38

Confidence: 52.06%; Support: 19.62%; Probability of consequent: 35.65%

5. Radius vs. Stay

   if age between 6 and 12, and admission year is between 1989 and 1992, and diagnosis is Radius, and operation is CR+POP, then stay is between 1 and 2 days.

   Fitness: 6.01

   Confidence: 81.11%; Support: 3.22%; Probability of consequent: 51.29%

   if diagnosis is Radius, and operation is CR+POP, then stay is between 1 and 2 days.

   Fitness: 5.49

   Confidence: 78.57%; Support: 10.22%; Probability of consequent: 51.29%

   if age is between 0 and 8, and diagnosis is Radius, then stay is between 0 and 3 days.

   Fitness: 2.89

   Confidence: 86.92%; Support: 10.19%; Probability of consequent: 71.30%

6. Humerus vs. Stay

   if diagnosis is Humerus, and operation is CR+K-WIRE, then stay is between 2 and 5 days.

   Fitness: 3.90

   Confidence: 67.30%; Support: 4.56%; Probability of consequent: 47.16%

7. Year vs. Stay

   if admission year is between 1985 and 1987, then stay is between 3 and 10 days.

   Fitness: 2.58

   Confidence: 46.98%; Support: 8.65%; Probability of consequent: 33.85%

A.4 The Best Rule Set Learned from the Scoliosis Database

A.4.1 Rules for Classification

King-I

1. if 1stMCGreater=N and 1stMCApex=T1-T8 and 2ndMCApex=L3-L4, then King-I.

   Fitness: 20.20

Confidence: 100%; Support: 0.86%; Probability of consequent: 28.33%

2. if 1stMCGreater=N and 1stMCDeg=21-80 and 1stMCApex =T1-T12 and 2ndMCApex=L2-L3, then King-I.

   Fitness: 19.06

   Confidence: 96.67%; Support: 6.22%; Probability of consequent: 28.33%

3. if 1stMCGreater=N and L4Tilt=Y and 1stMCApex =T1-T10 and 2ndMCApex=L2-L5, then King-I.

   Fitness: 18.92

   Confidence: 96.15%; Support: 10.73%; Probability of consequent: 28.33%

King-II

1. if 1stCurveT1=N and 1stMCGreater=Y and 1stMCDeg=16-45 and 2ndMCDeg=28-54 and 1stMCApex =T4-T11 and 2ndMCApex=L2-L3, then King-II.

   Fitness: 16.63

   Confidence: 100.00%; Support: 1.07%; Probability of consequent: 35.41%

2. if 1stMCGreater=Y and L4Tilt=Y and 1stMCDeg=22-77 and 2ndMCDeg=19-54 and 1stMCApex =T1-T11 and 2ndMCApex=L2-L2, then King-II.

   Fitness: 12.85

   Confidence: 87.88%; Support: 6.22%; Probability of consequent: 35.41%

3. if 1stMCGreater=Y and L4Tilt=Y and 1stMCApex=T6-T10 and 2ndMCApex= L2-L5, then King-II.

   Fitness: 10.52

   Confidence: 79.76%; Support: 14.38%; Probability of consequent: 35.41%

4. if 1stMajorCurveGreater=Y and 2ndMCDeg=8-95 and 1stMCApex=T3-T11 and 2ndMCApex= T4-T10, then King-II.

   Fitness: 3.32

   Confidence: 52.17%; Support: 7.73%; Probability of consequent: 35.41%

King-III

1. if 1stCurveT1=N and L4Tilt=N and 1stMCApex=T1-T9 and 2ndMCApex=Null, then King-III.

   Fitness: 5.87

   Confidence: 25.87%; Support: 0.86%; Probability of consequent: 7.94%

2. if L4Tilt=N and 1stMCApex=T2-T6 and 2ndMCApex=T2-T11, then King-III.

   Fitness: 4.86

   Confidence: 25.71%; Support: 1.93%; Probability of consequent: 7.94%

King-IV

1. if 1stCurveT1=Y and 1stMCGreater=Y and L4Tilt=Y and 1stMCApex=L5-T10 and 2ndMCApex=T9-L5, then King-IV.

   Fitness: 11.10

   Confidence: 29.41%; Support: 1.07%; Probability of consequent: 2.79%

2. if 1stMCGreater=Y and L4Tilt=Y and 1stMCApex=T10-L5 and 2ndMCApex=T5-L4, then King-IV.

   Fitness: 6.02

   Confidence: 19.35%; Support: 1.29%; Probability of consequent: 2.79%

King-V

1. if 1stMCGreater=Y and L4Tilt=Y and 1stMCApex=T2-T5 and 2ndMCApex=T9-T11, then King-V.

   Fitness: 22.75

   Confidence: 62.50%; Support: 1.07%; Probability of consequent: 6.44%

2. if 1stMCGreater=N and 2ndMCDeg=37-70 and 1stMCApex=T4-T7 and 2ndMCApex=T2-T11, then King-V.

   Fitness: 19.98

   Confidence: 57.14%; Support: 0.86%; Probability of consequent: 6.44%

3. if 1stCurveT1=Y and 1stMCGreater=Y and L4Tilt=Y and 1stMCDeg=3-35 and 1stMCApex=T2-T6 and 2ndMCApex=T7-T9, then King-V.

   Fitness: 16.42

   Confidence: 50.00%; Support: 0.86%; Probability of consequent: 6.44%

TL

1. if 1stMCGreater=Y and 1stMCApex=T11-T12 and 2ndMCApex=Null, then TL.

   Fitness: 19.49

   Confidence: 41.18%; Support: 1.50%; Probability of consequent: 2.15%

L

1. if 1stMCGreater=Y and L4Tilt=N and 1stMCApex=L2-L5 and 2ndMCApex=Null, then L.

   Fitness: 26.32

   Confidence: 62.50%; Support: 1.07%; Probability of consequent: 4.51%

2. if 1stCurveT1=N and L4Tilt=N and 2ndMCDeg=Null and 1stMCApex=L1-L3 and 2ndMCApex=Null, then L.

   Fitness: 21.59

   Confidence: 54.17%; Support: 2.79%; Probability of consequent: 4.51%

3. if 1stCurveT1=N and 1stMCApex=L2-L5 and 2ndMCApex=Null, then L.

   Fitness: 16.84

   Confidence: 45.45%; Support: 2.15%; Probability of consequent: 4.51%

A.4.2 Rules for Treatment

Observation

1. if Deg1=3-12 and Deg2 =Null and Deg3 = Null and Deg4 = Null, then Observation.

   Fitness: 7.59

   Confidence: 100.00%; Support: 1.93%; Probability of consequent: 62.45%

2. if Deg1=5-27 and Deg2 =4-21 and Deg3 = 0-22 and Deg4 = Null and mens = 99, then Observation.

   Fitness: 7.55

   Confidence: 100.00%; Support: 1.07%; Probability of consequent: 62.45%

3. if Deg1=4-13 and Deg2 =2-29 and Deg3 = Null and Deg4 = Null, then Observation.

   Fitness: 6.8

   Confidence: 95.55%; Support: 6.01%; Probability of consequent: 62.45%

Bracing

1. if age = 2-12 and Deg1=20-26 and Deg2 =24-47 and Deg3 = 27-52 and Deg4 = Null, then Bracing.

   Fitness: 22.54

   Confidence: 100.00%; Support: 0.86%; Probability of consequent: 24.46%

2. if Deg1=21-28 and Deg2 =32-43 and Deg3 = Null and Deg4 = Null and RI = 3-4, then Bracing.

   Fitness: 15.18

   Confidence: 80.00%; Support: 0.86%; Probability of consequent: 24.46%

3. if Deg1=25-39 and Deg2 =21-42 and Deg3 = Null and Deg4 = Null and RI = 1-3, then Bracing.

   Fitness: 12.26

   Confidence: 71.43%; Support: 1.07%; Probability of consequent: 24.46%

Appendix B

The Grammar used for the fracture and Scoliosis databases

B.1 The grammar for the fracture database

This grammar is not completely listed. The grammar for the other attribute descriptors is similar to the part of the grammar in lines 11-19.

1: *Rule* → *Rule1* | *Rule2* | *Rule3*
2: *Rule1* → if *Antes1* , then *Consq1* .
3: *Rule2* → if *Antes1* and *Antes2* , then *Consq2* .
4: *Rule3* → if *Antes1* and *Antes2* and *Antes3* , then *Consq2* .
5: *Antes1* → *Sex1* and *Age1* and *Admday1*
6: *Antes2* → *Diagnosis1*
7: *Antes3* → *Operation1* and *Surgeon1*
8: *Consq1* → *Diagnosis_descriptor*
9: *Consq2* → *Operation_descriptor* | *Surgeon_desriptor*
10: *Consq3* → *Stay_descriptor*
11: *Sex1* → any | *Sex_descriptor*
12: *Sex_descriptor* → sex = sex_const
13: *Admday1* → any | *Admday_descriptor*
14: *Admday_descriptor* → admday_day between day_const day_const
15: *Admday_descriptor* → admday_month between month_const month_const
16: *Admday_descriptor* → admday_year between year_const year_const
17: *Admday_descriptor* → admday_weekday between weekday_const weekday_const
18: *Diagnosis1* → any | *Diagnosis_descriptor*
19: *Diagnosis_descriptor* → diagnosis is diagnosis_const
...

B.2 The grammar for the Scoliosis database

This grammar is not completely listed. The grammar for the other attribute descriptors is similar to the part of the grammar in lines 7-12.

1: *Rule* → *Rule1* | *Rule2*
2: *Rule1* → if *Antes1* , then *Consq1* .
3: *Rule2* → if *Antes2* , then *Consq2* .
4: *Antes1* → *1stCurveT1* *1stMCGreater* and *L4Tilt* and *1stMCDeg* and *2ndMCDeg* and *1stMCApex* and *2ndMCApex*
5: *Antes2* → *Age* and *Lax* and *Deg1* and *Deg2* and *Deg3* and *Deg4* and *Mens* and *RI* and *TSI* and *ScoliosisType*
6: *Consq1* → *ScoliosisType_descriptor*
7: *1stMCGreater* → any | *1stMCGreater_descriptor*
8: *1stMCGreater_descriptor* → 1stMCGreater = boolean_const
9: *1stMCDeg* → any | *1stMCDeg_descriptor*
10: *1stMCDeg_descriptor* → 1stMCDeg between deg_const deg_const
11: *1stMCApex* → any | *1stMCApex_descriptor*
12: *1stMCApex_descriptor* →1stMCApex between Apex_const Apex_const
...