An algorithm which learns multiple covers via integer linear programming

Part I: the CLILP2 algorithm

Krzysztof J. Cios
University of Toledo, Ohio, USA and
Ning Liu
Dana Corporation, Ottawa Lake, Michigan, USA

Introduction

Learning a description of a concept from training examples is an inductive learning process. Owing to the nature of induction, there are many possible ways in which decision rules can be generated from learning examples. Algorithms for generating single hypotheses need mechanisms to determine what is the "best" description of the concept to be learned. Unfortunately, learning examples are usually insufficient to define a concept description uniquely. Even if one employs various heuristics, like decision-tree growing or if-then decision rule generation to classify learning examples, it does not necessarily mean that the generated description will reveal the "true" meaning of a concept. A description is only one of many possible interpretations of training data and may express meaning far different from that of a concept. Thus, the ability to generate multiple hypotheses from learning examples is an important aspect of inductive learning.

A major problem in designing learning algorithms is how to avoid generating excessively complex description from noisy examples. Learning from noise-corrupted data may result in a large number of complicated decision rules describing many trivial instances, and the resulting concept description may lack generality. This phenomenon is called "overfitting". It refers to a tendency to force the rule induced from training data to agree with these data too closely, at the expense of generalization to other examples. The other cause of overfitting may be due to a poor concept description language. How to deal with the noise-caused overfitting has been studied by machine learning

Editor's Note: This article should be read in conjunction with the authors' previous contribution, "Generation of diagnostic rules via inductive machine learning", *Kybernetes*, Vol. 22 No. 5, 1993, pp. 44-56. This work was partially supported by the NSF grant number DDM-9015333, the American Heart Association grant number NW-90-02, and the University of Toledo Research Support Program. The authors would also like to thank Dr Igor Mozetic for permission to use the lymphatic cancer, primary tumour, and breast cancer data.

researchers for a long time. There are two basic approaches to solving this problem. One is to allow a certain degree of inconsistent classification of training examples so that the descriptions will be general enough to describe basic characteristics of a concept. This approach has been taken by the ID family of algorithms[1-3]. The C4 algorithm of Quinlan[4], a direct descendant of ID3, converts its tree into rules and prunes both rule conditions and whole rules. The second approach is to discard unimportant rules and only retain those covering the largest number of examples and consider them as a general description of a concept. Typical algorithms using this technique are the AQ family of algorithms[5-7].

Retaining multi-meaning of training examples is not frequently addressed because it requires more complicated procedures, larger amounts of memory, and longer processing time. Although noise tolerance is a distinguished characteristic of a well-designed machine learning algorithm, in this article we shall emphasize the generation of multiple hypotheses by introducing an efficient learning algorithm. It is difficult to decide whether noise elimination or multiple interpretation of training examples is contributing more to finding a "better" concept description. Thus, using only noise-eliminating heuristics may not be sufficient to make such a decision. Let us illustrate the problem with the following example. There are ten training examples representing two classes. Each example is described by three features: shape, size, and colour of an object. The shape has five values: square, rectangle, box, ellipse, and circle. The values of size are: small, medium, and large. The values of colour are: white, black, and grey. The training and testing examples are depicted in Figure 1. A brief examination of examples leads to the following hypotheses (there are actually many more):

- IF the size of an object is not large THEN it belongs to class #1.
- IF the colour of an object is grey THEN it belongs to class #1.
- IF the shape of an object is not an ellipse and the size is not large THEN it belongs to class #1.

Since all the above hypotheses are equally effective in classifying the training examples, there is a problem as to which one should be generated and retained. In the ID3-type algorithms, the hypotheses are generated based on the maximum information principle. If one feature, such as size or colour, is detected to give the maximum information gain and classify training examples correctly, the learning process is terminated. Then, the third hypothesis will never be generated because the features like shape and size are not optimal in the sense of the maximum information principle. On the other hand, Clark and Niblett[8] proposed that the induced hypotheses should be as short as possible. This implies that the first two hypotheses would have higher priority to become a concept description. However, if we apply the above three hypotheses to the first set of test data, we notice that only the second hypothesis is correct. If we apply the hypotheses to the second set of test data, it is easy to notice that only

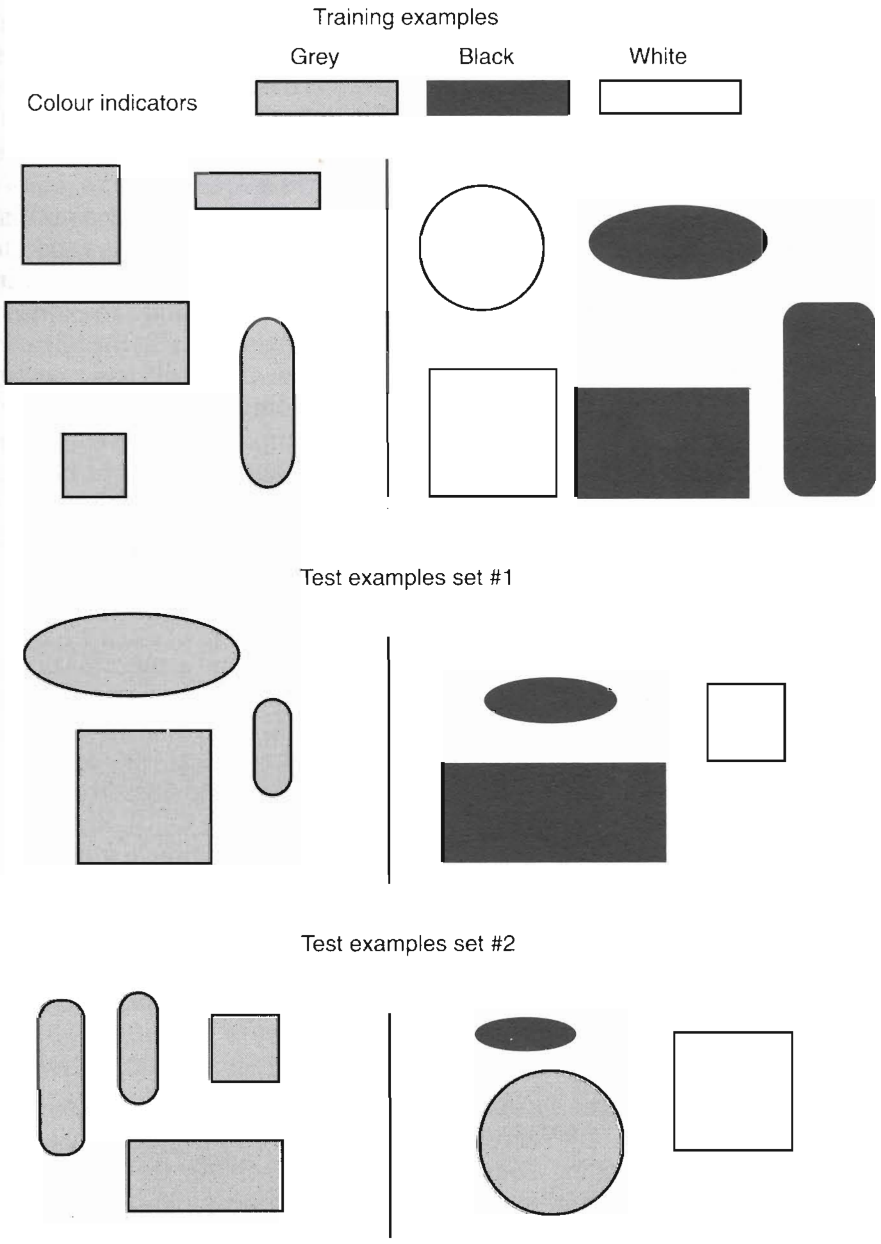

Figure 1. Training and test examples

the third hypothesis is correct. Overall, none of the generated hypotheses recognizes both sets of test data. To classify all test examples, a hypothesis that should have been generated is:

- IF the colour of an object is grey and the shape is not a circle THEN it belongs to class #1.

The above described "false" learning was not caused by noise, because learning examples were selected from a noise-free environment, but by the limited number of available training examples. In the learning process, it is assumed that a description of training data is sufficient as a concept description. However, training examples in real-world applications often convey information that can be interpreted in many different ways.

To address this problem, we shall present here a machine learning algorithm, CLILP2 (cover learning using integer linear programming), that generates multiple hypotheses of a concept in order to increase the chance of capturing the "true" meaning of a concept. At the same time, the algorithm is designed to solve the problem of overfitting while learning from noisy data. In the following sections, we first summarize two popular families of algorithms and then introduce the basic ideas of CLILP2. Next, the complexity of the CLILP2 algorithm will be analysed and an illustrative example is presented to show how the algorithm works. The algorithm is then applied to three types of medical data. The performance of CLILP2 is evaluated using two methods in order to compare our results with those obtained by others. Finally, the advantages of the algorithm are discussed.

Decision tree and rule-based algorithms

These algorithms are empirical learning algorithms that generate concept descriptions from examples by following specific procedures, and by using a set of heuristics in separating examples of one class from other classes. There are two major families of such algorithms: the decision tree-based algorithms, and the rule-based algorithms. An example of decision tree-based algorithms is the ID family of algorithms, e.g. ID3[1], ASSISTANT[2] and C4[3]. Rule-based algorithms are represented by the AQ family of algorithms, e.g. AQ11[9], AQ15[6] and AQTTT-15[7].

In this article, we shall present the CLILP2 algorithm that combines the advantages of both families of algorithms. First, we describe noise tolerance and multiple hypotheses generation capabilities of the two families. Readers may refer to Clark and Niblett[8] and Michalski[7] for more details.

Decision tree-based algorithms

Decision tree-based algorithms usually use the information entropy measure[1,2] to grow a decision tree by searching for a feature that gives maximum information gain. By dividing examples into smaller subsets, the procedure of growing a decision tree continues until the training examples are correctly classified according to a user-specified termination criterion. Although the ID3 algorithm[1] was originally designed to generate a concept description in a noise-free learning environment, its general-to-specific search leads to an easy modification using the tree-pruning techniques[10-12] to avoid overfitting effectively while learning from noise-corrupted data. A probabilistic approach was also proposed[3]. Extension of the ID3 algorithm, continuous ID3 (CID3) algorithm was developed[13] to self-generate a neural network architecture.

However, the problem of generating multiple hypotheses from training examples is not frequently addressed. In real-world applications training examples are usually insufficient to define a concept description uniquely. Therefore, learning algorithms need a flexibility to produce different generalizations from given examples. As illustrated by the example in the last section, one may obtain misleading descriptions of a concept regardless of whatever search heuristics are used. In such a case, an algorithm generating multiple hypotheses has a better chance of avoiding such "false" descriptions. In decision tree-based algorithms, the description of a subset of examples in a leaf node of a tree is uniquely described as series of feature tests from the root to the bottom of a tree. This approach lacks the flexibility of describing a target concept in different ways.

Rule-based algorithms
Rule-based algorithms have the capability of generating multiple descriptions of a concept. An example is the AQ15 algorithm where the empirical learning was treated by Michalski[5] as the general covering problem. The basic term of a *cover*, used in the AQ family of algorithms, implies that there may exist multiple covers to cover positive training examples. Actually, the search for a minimum number of complexes to cover all positive examples is NP-complete (a complex constitutes a cover, and is a description of a subset of positive examples). This resulted in the development of procedures that produce a quasi-optimal solution in polynomial time. Generally, AQ algorithms follow a greedy heuristics that tries to include/exclude as many as possible of positive/negative examples in searching for a complex. The AQ algorithms employ a set of user specified description preference criteria to describe a subset of positive examples covered by a complex. Because of the flexibility in describing concepts in different ways by using VL1 logic expressions[14], the AQTT-15[7] generates the "best" descriptions as the base concept representation and employs flexible matching of concepts.

Although the AQ algorithms allow a cover to be generated in different ways by selecting, either in random or in a predesigned order, a set of particular positive examples called seeds, only one way of partitioning positive training examples is obtained each time. In searching for a different partition schema, some complexes covering a small set of particular examples shall always be included for all covers, while some complexes covering a larger number of examples may be included in one cover but not in other covers. Thus, a different partition determines how a target concept is described. Although AQ algorithms are capable of generating multiple descriptions, those descriptions are limited to describing one partition schema.

Procedures adopted to solve the general covering problem in AQ algorithms allow for generation of different covers, but the requirement to cover all positive examples restricts the application of noise-elimination heuristics. AQTT-15 algorithm[6] has left the basic AQ algorithm intact and introduced postprocessing techniques such as the TRUNC procedure, TT-representation, and flexible matching[7] to cope with noisy data.

CLILP2 algorithm

The objective of the CLILP2 algorithm is to combine the advantages of the ID and the AQ families of algorithms to generate multiple hypotheses that do not "overfit" target concepts. More specifically, CLILP2 adopts the "tree-pruning" technique from the ID3 family of algorithms and the idea of cover generation from the AQ family of algorithms. CLILP2 partitions training examples into "noise-free" subsets and then generates covers from the quasi-optimal solutions of a general covering problem. CLILP2 generates concept descriptions by solving integer linear programming (ILP) models. The learning process consists of two phases. In phase I, the positive examples are partitioned into subsets in almost every possible way so that one will have an opportunity to select the "best" complexes for a cover. To prevent the drastic increase in the number of subsets, a noise-elimination procedure similar to the "tree-pruning" is introduced so that the partition of positive examples results in a relatively small number of subsets. In phase II, several covers that consist of the minimum number of complexes are obtained by solving one ILP model, and then multiple descriptions for each complex are generated by solving another ILP model. The CLILP2 algorithm follows:

```
(1) Learning phase I:
    ● Let POS be the set of $M$ positive examples.
      Let NEG be the set of $N$ negative examples.
    ● Partition procedure:
      Let COMPLEXES be the empty list.
      Let POS be complex$_0$ that contains all positive examples.
      Repeat ($N$ iterations) for each negative example:
      – Select a negative example, neg$_i$, from NEG and compare it with
        all subsets, denoted as complex$_j$ ($j = 1, 2, 3, \ldots$).
      – Divide the subsets into smaller ones so that positive examples in
        each new subset have at least one feature value different from
        features of neg$_i$.
      – Delete redundant subsets using noise-eliminating heuristics
        (explained later).
      – Delete/add old complex$_j$/new complex$_k$ from/to COMPLEXES.
(2) Learning phase II:
    ● Let COVERS be the empty list.
    ● Convert COMPLEXES into constraints of the first ILP model and
      add the multiple solutions as hypotheses into COVERS.
    ● For all hypotheses in COVERS, repeat:
      – For complex$_i$ from COVERS.
```

- Repeat for all covered positive examples.
- Compare pos$_i$ with NEG. If a negative example has some features identical to that of pos$_i$, change them to –1.
- Convert NEG into the constraints of the second ILP model and obtain descriptions of the target concept in the form of if-then rules.

The objective of dividing positive examples into subsets is to search for all possible ways of separating them from negative examples. At first, all positive examples are compared with all negative examples. Then, after the first negative example is compared with all positive examples, and the positive examples having at least one feature different from those of the negative example form a subset. In this way, all possible groupings of positive examples, against that negative example into subsets, are obtained. When the second negative example is now compared with all the subsets, the newly created subsets show all the possible ways of separating the positive examples from the first two negative examples. As this process continues, the final division of positive examples results in all the possible ways of separating them from all the negative examples.

This N step comparison allows for various divisions of positive examples, and for elimination of the embedded noise. If the training examples are noisy, the division of positive examples usually results in a large number of similar subsets. For instance, let us consider a subset consisting of examples *a, b, c, d, e*, and *f*, denoted as (*a, b, c, d, e, f*). There may exist, however, other subsets such as (*a, b, c, d, e*), (*a, b, c, d, f*), or (*a, c, d, e, f*). The slight difference between the subsets may have been caused by noise. If the difference is not significant, it is reasonable to discard some subsets and keep only one subset which can be considered as a noise-free. Thus, the new division of positive examples usually starts from a reduced number of subsets. This technique is similar to tree-pruning.

Because the partition of positive examples includes every possible division, CLILP2 selects only the “best” complexes which cover all positive examples. In this work, we assume that noise is causing the difference between subsets so that elimination of severely distorted subsets will not preclude us from finding complexes covering all positive examples.

Several preference criteria can be used to generate a cover: the *largest complex first, fit first*, or *background knowledge first*. Large complexes are those covering many positive examples, and they can be ranked according to the number of examples covered. It should be noted, however, that a group of large complexes may not cover all positive examples in terms of the minimum number of complexes. The *fit first* preference used by CLILP2 allows it to search for the best match among complexes (large or small) to cover all examples using minimum number of complexes. Using the *background knowledge* preference, if some positive examples are thought to be more typical than others, then the complexes covering those examples have higher priority for inclusion in a cover. This preference criterion, also used by CLILP2, is particularly useful for incremental learning without the need to remember past examples.

To satisfy the requirement of covering all positive examples and to generate covers using different preference criteria, an integer linear programming model is built to generate various solutions. In the ILP model, each variable corresponds to a complex: $x_i = 1$ means complex$_i$ is included in a cover; $x_j = 0$, means it is not. The weights of the variables in the objective function are determined by the cover generation preference. A solution of this ILP indicates which complexes are to be included in a cover. CLILP2 obtains multiple solutions which serve as multiple hypotheses for description of a concept.

It is interesting to note that in learning phase II the if-then rules are not generated until a complex is selected for a cover. This generation of concept descriptions is different from those used in decision tree- and rule-based algorithms which simultaneously generate concept description and partition training examples. Since CLILP2 generates more complexes than it finally uses for a cover, there is no need to generate descriptions for the complexes not used. Thus the cost, in terms of memory space and computing time, of generating a large number of complexes is low since information about the examples included in each subset is stored in one matrix.

CLILP2 not only generates different covers, as multiple hypotheses for a concept, but it is also capable of generating multiple descriptions for each complex. This means that different if-then rules can be generated as a final description of a concept. By solving the second ILP model, different descriptions of positive examples covered by a complex can easily be obtained. Unlike the AQ family of algorithms that use a rule preference criterion for generating a concept description, CLILP2 measures the frequency of a feature appearing in multiple solutions to determine the importance of an if-then rule. The more frequently a feature appears in different solutions, the more likely it describes the basic characteristic of a concept. The features with high frequencies are called key features. Thus, rules describing a concept are ranked by the average frequency of features in that rule. The top-ranking rule contains the largest number of key features. If key features are at a tie and appear in different rules, one may synthesize a new rule by combining key features from different rules. This approach results in a single if-then rule that summarizes all basic characteristics as the description of a concept.

Complexity of CLILP2

In order to evaluate the complexity of CLILP2, let us denote the number of positive training examples by n_{pos}, and the number of negative training examples by n_{neg}. Each example is described by k features.

In learning phase I, a decision tree is built and trimmed. This requires the following operations:

- Selecting a negative example and comparing it with all the positive examples: it takes time $O(n_{\text{pos}} * k)$.

examples: it takes time $O(n_{\text{pos}} * k)$.

- Updating complex matrix from the previous iteration: it takes time $O(n_{\text{pos}} * k^2)$.
- Simplifying the complex matrix by deleting noise-distorted subsets. This takes time $O(w * k^2)$, where w is the width of the decision tree at a current level L.

The upper bound of a maximum tree width is $m * n_{\text{pos}}$ where m is a large integer used to trim a decision tree before it becomes too "bushy", so we have: $w < m * n_{\text{pos}}$. The time taken in this step is: $O(m * n_{\text{pos}} k^2)$.

- The operations of the above three steps are repeated for all negative examples. Overall, the learning phase I takes time: $O(n_{\text{neg}} n_{\text{pos}} * k^2)$ or $O(n^2 * k^2)$, where $n = n_{\text{pos}} + n_{\text{neg}}$.

In learning phase II, the final complex matrix is used to construct the ILP model. The size of the model is measured as $m * n_{\text{pos}} * k$. It has been shown by Bar Yehuda and Even[15] that there exists an algorithm to find approximate solution in polynomial time, $O(n)$, where n is the size of the problem. Therefore, the time to solve our problem is $O(n_{\text{pos}} * k)$. To derive decision rules from each selected complex, a set-covering problem is solved again. This takes time $O(n_{\text{neg}} * k)$. The overall time complexity is then $O(n_{\text{pos}} * k + n_{\text{neg}} * k)$, so the complexity of CLILP2 is then evaluated as:

$$O(n^2 * k^2).$$

Finally, we shall stress that the complexity of many machine learning algorithms heavily depends on the heuristics used. In many cases, extensive trimming of a decision tree will solve a problem in a shorter time but at the expense of accuracy.

Illustration of CLILP2 operation

To illustrate further the basic ideas of the CLILP2 algorithm, we shall explain them by means of an example using a subset of data previously utilized in other work[16]. Our training example is described by ten features. The values of each feature can take on values from 1 to 9. There are 106 training examples of three different classes. In order to induce concepts for class #1, all 18 examples belonging to the class are stored in matrix POS, and the 88 examples of other classes are stored in matrix NEG. The entire matrix POS is shown in (1) together with only a few examples from matrix NEG because of its large size.

Learning phase I

In this phase, two major operations are performed: partition of the positive examples and elimination of subsets that are corrupted with noise. Since the partition process is similar to the process of growing a search tree, we use a leaf

$$
\mathrm{POS} = \begin{bmatrix}
4&4&3&4&4&3&3&2&2&2\\
5&5&5&5&5&5&4&4&4&4\\
5&5&5&5&5&3&3&5&4&2\\
2&4&4&4&4&4&4&3&4&3\\
2&1&4&4&2&4&1&1&1&2\\
3&3&3&3&4&4&3&3&3&2\\
5&5&5&5&5&5&9&5&9&5\\
4&4&5&4&4&5&3&4&3&4\\
4&5&4&3&3&4&4&5&4&4\\
2&2&4&2&2&2&2&2&4&4\\
4&4&3&4&4&3&4&3&3&3\\
4&5&4&4&2&3&2&1&1&4\\
2&3&2&2&2&2&3&4&2&3\\
4&5&3&4&2&2&3&4&2&3\\
1&2&5&5&5&5&4&1&3&5\\
4&4&3&4&3&3&3&4&3&3\\
4&3&4&3&4&3&4&3&3&3\\
3&3&3&3&4&3&3&3&3&3
\end{bmatrix}_{18\times 10}
\qquad
\mathrm{NEG} = \begin{bmatrix}
4&5&5&4&5&5&5&3&5&5\\
4&5&4&4&4&5&4&5&4&4\\
5&4&5&5&5&5&5&5&4&5\\
4&4&5&4&4&5&5&5&5&4\\
\cdot&&&&&&&&&\\
\cdot&&&&&&&&&\\
\cdot&&&&&&&&&\\
4&4&3&4&4&3&3&4&4&4\\
\cdot&&&&&&&&&\\
\cdot&&&&&&&&&\\
\cdot&&&&&&&&&\\
5&4&5&5&5&5&5&3&5&4
\end{bmatrix}_{88\times 10}
\qquad (1)
$$

node to represent a subset, and a tree branch to represent further division of a subset. Correspondingly, the two operations are termed as the tree expansion and trimming of its branches.

Tree expansion

All positive examples are stored at the root of a search tree and the first operation is to choose a negative example, say neg_i, from matrix NEG and compare it with all examples from matrix POS (pos_j, = 1, 2, 3, …, 18). If some features of a positive example, pos_j, have the same values as the features of neg_i, then they cannot be used to distinguish between pos_j from neg_i. They are then changed to a –1 in the matrix POS. Matrix POS(neg_1), is obtained by comparing every positive example from matrix POS with the first negative example neg_1. It is shown in (2).

The next operation is to form the leaf nodes of the search tree. If we look at column 7 of matrix POS(neg_1) there are no –1s, which means that feature seven may be used to discriminate all positive examples from neg_1 (column 9 could have been used as well). The first feature test for separating all the positive examples from neg_1 is < F7 = 1, 2, 3, 4, 9 >. As a result, a vector called the complex vector is created and is also shown in (3). To record the fact that all the positive examples are moved into the first leaf node, a value 1 is used to indicate that the corresponding example belongs to the node. A value 0 indicates that it does not. Therefore a leaf node has been added to the root node.

$$POS(neg_1) = \begin{bmatrix} -1 & 4 & 3 & -1 & 4 & 3 & 3 & 2 & 2 & 2 \\ 5 & -1 & -1 & 5 & -1 & -1 & 4 & 4 & 4 & 4 \\ 5 & -1 & -1 & 5 & -1 & 3 & 3 & 5 & 4 & 2 \\ 2 & 4 & 4 & -1 & 4 & 4 & 4 & -1 & 4 & 3 \\ 2 & 1 & 4 & -1 & 2 & 4 & 1 & 1 & 1 & 2 \\ 3 & 3 & 3 & 3 & 4 & 4 & 3 & -1 & 3 & 2 \\ 5 & -1 & -1 & 5 & -1 & -1 & 9 & 5 & 9 & -1 \\ -1 & 4 & -1 & -1 & 4 & -1 & 3 & 4 & 3 & 4 \\ -1 & -1 & 4 & 3 & 3 & 4 & 4 & 5 & 4 & 4 \\ 2 & 2 & 4 & 2 & 2 & 2 & 2 & 2 & 4 & 4 \\ -1 & 4 & 3 & -1 & 4 & 3 & 4 & -1 & 3 & 3 \\ -1 & -1 & 4 & -1 & 2 & 3 & 2 & 1 & 1 & 4 \\ 2 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 2 & 3 \\ -1 & -1 & 3 & -1 & 2 & 2 & 3 & 4 & 2 & 3 \\ 1 & 2 & -1 & 5 & -1 & -1 & 4 & 1 & 3 & -1 \\ -1 & 4 & 3 & -1 & 3 & 3 & 3 & 4 & 3 & 3 \\ -1 & 3 & 4 & 3 & 4 & 3 & 4 & -1 & 3 & 3 \\ 3 & 3 & 3 & 3 & 4 & 3 & 3 & -1 & 3 & 3 \end{bmatrix}_{18\times10} \quad (2)$$

$$COM_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} 1\times18 \quad (3)$$

Next, another negative example, say neg_2, is selected and compared with all the examples of the original matrix POS. Looking at each column of the matrix $POS(neg_2)$, shown in (4), the examples not changed to –1 are included in one subset. Since there are ten columns (ten features) in the matrix POS, the maximum number of new leaf nodes is also ten. It may happen that all the positive examples of one column may appear in other columns as well. Then, this subset (column) is redundant. For instance, the examples of column 2 in matrix (4) constitute a redundant subset since all the examples in that subset are present in column 8. After deleting this redundant subset, there are nine possible ways of generating new leaf nodes, which is depicted in Figure 2. The corresponding features are shown on the branches indicating the leaf nodes' identities. In the matrix shown in (5), nine complexes are shown as nine row vectors. The search tree is used to illustrate the size of subsets, and the complex matrix indicates how positive examples are covered. A complex from matrix in (5) corresponds to a leaf node depicted in Figure 2.

$$\mathrm{POS}(\mathrm{neg}_2) = \begin{bmatrix} -1 & 4 & 3 & -1 & -1 & 3 & 3 & 2 & 2 & 2 \\ 5 & -1 & 5 & 5 & 5 & -1 & -1 & 4 & -1 & -1 \\ 5 & -1 & 5 & 5 & 5 & 3 & 3 & -1 & -1 & 2 \\ 2 & 4 & -1 & -1 & -1 & 4 & -1 & 3 & -1 & 3 \\ 2 & 1 & -1 & -1 & 2 & 4 & 1 & 1 & 1 & 2 \\ 3 & 3 & 3 & 3 & -1 & 4 & 3 & 3 & 3 & 2 \\ 5 & -1 & 5 & 5 & 5 & -1 & 9 & -1 & 9 & 5 \\ -1 & 4 & 5 & -1 & -1 & -1 & 3 & 4 & 3 & -1 \\ -1 & -1 & -1 & 3 & 3 & 4 & -1 & -1 & -1 & -1 \\ 2 & 2 & -1 & 2 & 2 & 2 & 2 & 2 & -1 & -1 \\ -1 & 4 & 3 & -1 & -1 & 3 & -1 & 3 & 3 & 3 \\ -1 & -1 & -1 & -1 & 2 & 3 & 2 & 1 & 1 & -1 \\ 2 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 2 & 3 \\ -1 & -1 & 3 & -1 & 2 & 2 & 3 & 4 & 2 & 3 \\ 1 & 2 & 5 & 5 & 5 & -1 & -1 & 1 & 3 & 5 \\ -1 & 4 & 3 & -1 & 3 & 3 & 3 & 4 & 3 & 3 \\ -1 & 3 & 4 & 3 & -1 & 3 & -1 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & -1 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}_{18\times 10} \qquad (4)$$

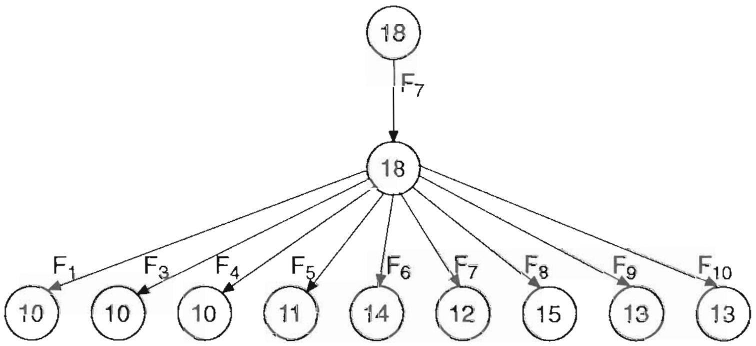

Figure 2.
Second level of the search tree

$$\mathrm{COM}_2 = \begin{bmatrix} 0&1&1&1&1&1&1&0&0&1&0&0&1&0&1&0&0&1 \\ 1&1&1&0&0&1&1&1&0&0&1&0&1&1&1&1&0&1 \\ 0&1&1&0&0&1&1&0&1&1&0&0&1&0&1&0&1&1 \\ 0&1&1&0&1&0&1&0&1&1&0&1&1&1&1&1&0&0 \\ 1&0&1&1&1&1&0&0&1&1&1&1&1&1&0&1&1&1 \\ 1&0&1&0&1&1&1&1&0&1&0&1&1&1&0&1&0&1 \\ 1&1&0&1&1&1&0&1&0&1&1&1&1&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1&0&0&1&1&1&1&1&1&1&1 \\ 1&0&1&1&1&1&1&0&0&0&1&0&1&1&1&1&1&1 \end{bmatrix}_{9\times18} \qquad (5)$$

Since the number of leaf nodes may increase rapidly as more negative examples are used for comparisons, the next section introduces the search tree-trimming technique in order to keep the search tree relatively small. We have to make sure, however, that branch trimming will not cause serious information loss.

Trimming search tree branches
Complex matrix COM_2 stores all possible non-redundant complexes for separating all positive examples from the first two negative examples. By comparing these nine complexes stored in COM_2, one may notice that some vectors are similar. For instance, complex #8 (eighth row vector) is close to complex #9. This similarity can be explained in two ways.

First, this difference between the complexes may have been caused by noise. Let us consider a complex covering some positive examples obtained from a noise-free environment. Each covered example may be thought of as a small particle randomly located in a circle as shown in Figure 3. Because of the measurement errors, some feature values are corrupted with noise, and the examples, represented as particles, get enough energy to move out and thus will not be covered by the complex, while some outside particles will enter the circle. If we assume that the movement of a particle varies with different noise levels, then the complexes can be viewed as an instrument recording the movement of

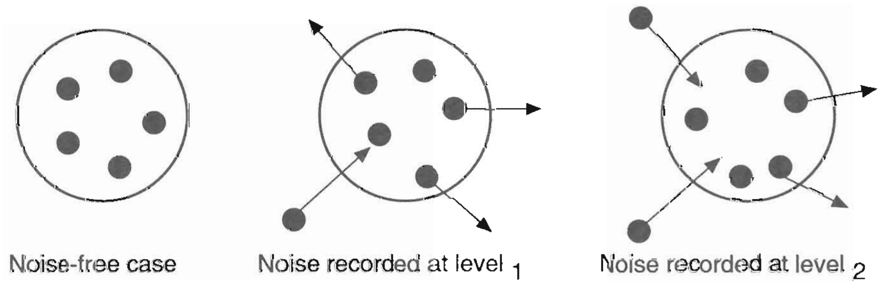

Figure 3. The examples covered by different complexes

particles at different noise levels. In conclusion, we may say that similar complexes represent generally the same information about the concept to be learned. The noise-elimination strategy used makes sure that only complexes which cover the largest number of examples are retained.

Another possible interpretation of the overlap between the complexes is that individual complexes actually do convey different meanings of a concept. In this case, the difference should be retained. However, if the difference is very small, it is still worthwhile to trade off a slight information loss for a smaller-size search tree.

Summarizing, while dividing positive examples into groups, the difference between the obtained subsets should be significant. If not, the similar subsets are discarded. This approach provides a good mechanism for eliminating the impact of noise. The subset (complex) covering the largest number of positive examples is called a "noise-free subset" and the other subsets are called "noise-distorted subsets". The corresponding leaf nodes in a search tree are then said to be noise-free nodes or noise-distorted nodes. If one selects, however, the largest subset from a group of similar subsets as a starting-point for the division process, the number of possible divisions of positive examples is greater than selecting a smaller subset as a starting-point. To this end, a branch-trimming heuristic is introduced to eliminate noise-distorted nodes. CLILP2 uses an arbitrary noise-distortion threshold, for instance 30 per cent, and calculates the percentage of overlap between any two complexes. If a complex is identified as similar to (more than 30 per cent) and smaller than other complexes, it is identified as a noise-distorted complex and will be deleted. Before a noise-distorted node is deleted, a test is performed to make sure that the remaining complexes still cover all the positive examples.

Although it has been a popular belief that consistent classification of training examples lacks flexibility in handling noise, the noise-elimination approach adopted in CLILP2 demonstrates that the conflict between consistent classification and noise elimination can be decreased, if not totally eliminated. By varying the noise-distortion threshold one may easily determine the impact of noise on the training examples. When examples are heavily corrupted with noise, the division of positive examples is sensitive to adopted thresholds.

In the above example, after the elimination of noise-distorted nodes from the search tree using a threshold of 30 per cent (chosen arbitrarily), only three leaf nodes will be expanded. The resulting decision tree is shown in Figure 4, and the corresponding complex matrix is shown in (6).

$$\mathrm{COM}_2 = \begin{bmatrix} 1&0&1&1&1&1&0&0&1&1&1&1&1&1&0&1&1&1 \\ 1&1&0&1&1&1&0&1&0&1&1&1&1&1&1&1&1&1 \\ 1&0&1&1&1&1&1&0&0&0&1&0&1&1&1&1&1&1 \end{bmatrix}_{3\times 18} \qquad (6)$$

Finally, after all negative examples have been tested, the 18 positive training examples are divided into nine subsets. The results are shown in the matrix COM_{88} (7).

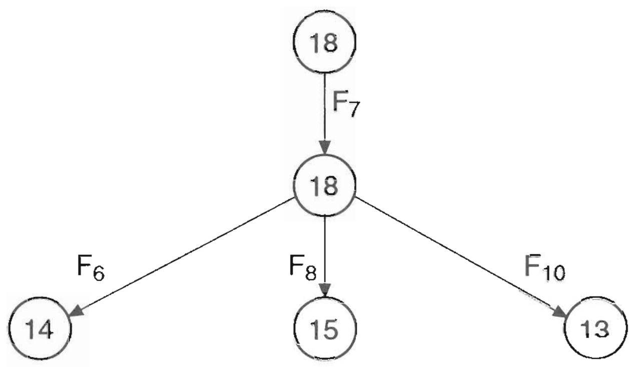

Figure 4. The search tree after trimming

Learning phase II
In this phase, covers are generated from the resultant complex matrix COM_{88} as descriptions of a concept. For each complex selected in forming a cover, the corresponding decision rule is induced by solving an integer linear programming model.

Cover formation
CLILP2 builds the covers according to the user's preference. The *largest complex first* preference allows the inclusion of complexes covering a large number of positive examples. When background knowledge that some examples are more typical for learning a concept is available, the complexes covering those examples are assigned a higher priority and will be included in a cover; it is called the *background knowledge first* preference. By default, CLILP2 uses *fit first* preference and searches for a minimum number of complexes for a cover.

From the complex matrix COM_{88}, the ILP model is built. The variable x_i in the objective function corresponds to a complex. The priorities of complexes are represented by weights in the objective function. While using the *largest complex first* preference, the weights are calculated according to the number of

$$COM_{88} = \begin{bmatrix} 0&1&0&0&1&0&0&0&0&1&0&1&1&0&1&0&0&0 \\ 0&0&0&0&1&0&0&1&0&1&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&0&0&0&0&0&0&1&1&0&1&0&1 \\ 0&0&1&1&1&1&0&0&0&0&1&0&0&0&0&1&1&1 \\ 0&0&0&0&1&1&1&0&0&0&0&0&1&0&1&0&0&1 \\ 1&0&0&0&0&0&1&0&0&0&1&0&0&1&1&1&0&0 \end{bmatrix}_{9\times18} \qquad (7)$$

examples covered. The weights for the *background knowledge first* preference are calculated according to the number of existing rules covered. When using default *fit first* preference all complexes are treated as equally important. The objective function is minimized to find the minimum number of complexes for a cover. In order to satisfy the constraint that every positive example is covered, all constraints are in "≥" form and the right-hand side values are all 1. The constraint coefficients are obtained by transposing the complex matrix (7). Model (8) shows how the covers are generated by using the *fit first* preference. This integer linear programming problem is known as the set-covering problem[17,18].

$$
\begin{array}{l}
\min\ Z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \\
\text{subject to} \\
\begin{array}{cccccccccccccccccc}
&&&&&&&&&& x_6 &&&&&& x_9 & \geq 1 \\
x_1 &&&&&&&&&&&&&&&&& \geq 1 \\
&&&&&&&&&& x_6 & + & x_7 &&&&& \geq 1 \\
&&&&&&&&&&&& x_7 &&&&& \geq 1 \\
x_1 & + & x_2 & + &&&&&&& x_6 & + & x_7 & + & x_8 &&& \geq 1 \\
&&&&&&&& x_5 & + &&& x_7 & + & x_8 &&& \geq 1 \\
&&&&&&&&&&&&&& x_8 & + & x_9 & \geq 1 \\
&& x_2 &&&&&&&&&&&&&&& \geq 1 \\
&&&& x_3 & + &&& x_5 &&&&&&&&& \geq 1 \\
x_1 & + & x_2 & + &&& x_4 & + & x_5 &&&&&&&&& \geq 1 \\
&&&&&&&&&&&& x_7 & + &&& x_9 & \geq 1 \\
x_1 & + &&&&&&&&&&&&&&&& \geq 1 \\
x_1 & + & x_2 & + &&&&&&& x_6 & + &&& x_8 &&& \geq 1 \\
&&&&&&&&&& x_6 & + &&&&& x_9 & \geq 1 \\
x_1 & + &&&&&&&&&&&&& x_8 & + & x_9 & \geq 1 \\
&&&&&&&&&& x_6 & + & x_7 &&&& x_9 & \geq 1 \\
&&&& x_3 & + & x_4 & + &&&&& x_7 &&&&& \geq 1 \\
&&&&&&&&&& x_6 & + & x_7 & + & x_8 &&& \geq 1
\end{array}
\end{array}
\tag{8}
$$

Because the set-covering problem is NP-complete, we shall look for an approximate solution as the description of the concept. Fortunately, there exist polynomial algorithms for finding the approximate solutions of this ILP model[15,19,20]. In this work, Bar Yehuda and Even's algorithm[15] is utilized to generate multiple covers (see Appendix). Several approximate solutions are found from this ILP model. The maximum number of solutions is specified by the user. The covers generated in this manner are called hypotheses. Matrix (9) shows three hypotheses generated from the ILP model (8).

$$\mathrm{SOL} = \begin{bmatrix} 1\,1\,1\,0\,0\,0\,1\,0\,1 \\ 1\,1\,0\,0\,1\,0\,1\,0\,1 \\ 1\,1\,1\,0\,0\,1\,1\,1\,0 \end{bmatrix}_{3\times 9} \quad (9)$$

Each solution specifies the complexes to be included in a hypothesis. For example, the first solution (first row) contains complexes #1, #2, #3, #7 and #9. Here, each element in a column of matrix (9) serves as an index of complexes (row vectors) of matrix (7). From each complex, a decision rule is formed, which is explained in the next section.

Decision rule formation

The description of a complex is not specified until the complex is selected as a part of the cover. To specify a complex, that is to build selectors from the covered examples, the common features are utilized. By a common feature we mean that a feature, with possibly more than one value, is present in the group of positive examples and is not present in all negative examples. For instance, even though positive examples 14 and 15 from matrix POS (1) share no identical feature values, they are covered by complex 9. Thus, the common feature is identified as F_9 that takes on the values 2, 3, and 9. The feature test is built as < F9 = 2, 3, 9 >. From a set of positive examples one usually can generate different decision rules. Some features will be called *key features* because they appear in most of the generated rules, while the other features are called *auxiliary features* because they appear in most of the generated rules. If an example satisfies every *key feature* but not the *auxiliary features*, it will be very likely covered by that complex. To generate reliable decision rules, one needs a mechanism to determine whether a feature is a *key feature* or an *auxiliary* one. To find the key features we shall solve the second ILP model.

The constraints for the ILP model are constructed by collecting the covered positive examples and comparing them with all the negative examples from matrix NEG. To generate a decision rule, for instance, from complex 9 in matrix SOL (9), the covered examples are selected from matrix POS and stored in a new matrix POS(com_9) (10). The covered examples are then #1, #7, #11, #14, #15 and #16.

$$\mathrm{POS}(\mathrm{com}_9) = \begin{bmatrix} 4\,4\,3\,4\,4\,3\,3\,2\,2\,2 \\ 5\,5\,5\,5\,5\,5\,9\,5\,9\,5 \\ 4\,4\,3\,4\,4\,3\,4\,3\,3\,3 \\ 4\,5\,3\,4\,2\,2\,3\,4\,2\,3 \\ 1\,2\,5\,5\,5\,5\,4\,1\,3\,5 \\ 4\,4\,3\,4\,3\,3\,3\,4\,3\,3 \end{bmatrix}_{6\times 10} \quad (10)$$

Next, a positive example from matrix POS(com_9) is selected and compared with matrix NEG. This operation is similar to building matrix POS(neg_i) as described earlier. If a feature of a negative example has the value identical to that of a positive example in POS(com_9), then this feature in NEG is changed to –1, meaning that it will not be used. After the first positive example from matrix POS(com_9) is compared with matrix NEG, matrix NEG is changed to matrix NEG(pos_1). Part of the matrix is shown in (11).

$$\mathrm{NEG}(\mathrm{pos}_1) = \begin{bmatrix} -1 & 5 & 5 & -1 & 5 & 5 & 5 & 3 & 5 & 5 \\ -1 & 5 & 4 & -1 & -1 & 5 & 4 & 5 & 4 & 4 \\ 5 & -1 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 5 \\ -1 & -1 & 5 & -1 & -1 & 5 & 5 & 5 & 5 & 4 \\ & & & & \vdots & & & & & \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 4 & 4 & 4 \\ & & & & \vdots & & & & & \\ 5 & -1 & 5 & 5 & 5 & 5 & 5 & 3 & 5 & 4 \end{bmatrix}_{88\times 10} \tag{11}$$

Next, another positive example from POS(com_9) is selected and compared with the examples from matrix NEG(pos_1). The comparison is the same as before except that the previously obtained 1s remain unchanged. The final matrix NEG(pos_{16}) is shown in (12) from which the constraints for the second ILP model are obtained.

$$\mathrm{NEG}(\mathrm{pos}_{16}) = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & 5 & -1 & 5 & -1 \\ -1 & -1 & 4 & -1 & -1 & -1 & -1 & -1 & 4 & 4 \\ -1 & -1 & -1 & -1 & -1 & -1 & 5 & -1 & 4 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 5 & -1 & 5 & 4 \\ & & & & \vdots & & & & & \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 4 & 4 \\ & & & & \vdots & & & & & \\ -1 & -1 & -1 & -1 & -1 & -1 & 5 & -1 & 5 & 4 \end{bmatrix}_{88\times 10} \tag{12}$$

The formation of the second ILP model is like one already described. In this ILP model, a variable x_i is used to represent a feature to be chosen for a decision rule. The coefficients for constraints are taken from matrix NEG(pos_{16}) where –1 corresponds to coefficient 0 and values other than –1 are changed into coefficients of 1. From the ILP model (13), several approximate solutions are obtained. The difference between an approximate solution and an optimal solution lies in the *auxiliary features* which do not have much influence on inducing the concept description. From the single optimal solution we could not specify whether a feature is a *key feature* or an *auxiliary* one. A group of approximate solutions, however, clearly shows the difference between the features. Four approximate solutions with arbitrarily selected rule-truncation threshold (0.90) are shown in (14).

$$\begin{aligned}
&\min\ Z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}\\
&\text{subject to:}\\
&\qquad x_7 + x_9 \ge 1\\
&\qquad x_3 + x_9 + x_{10} \ge 1\\
&\qquad x_7 + x_9 \ge 1\\
&\qquad x_7 + x_9 + x_{10} \ge 1\\
&\qquad \vdots\\
&\qquad x_9 + x_{10} \ge 1\\
&\qquad \vdots\\
&\qquad x_7 + x_9 + x_{10} \ge 1
\end{aligned} \tag{13}$$

$$\mathrm{SOL}_2 = \begin{bmatrix} 0\,1\,1\,0\,0\,0\,1\,0\,1\,1\\ 1\,1\,1\,0\,0\,0\,1\,0\,1\,0\\ 1\,1\,0\,0\,0\,1\,1\,0\,1\,0\\ 0\,1\,0\,0\,0\,1\,1\,0\,1\,1 \end{bmatrix}_{4\times 10} \tag{14}$$

The frequency with which a feature appears in the solution may be treated as an indicator of its importance. In the solution matrix (14), features 2, 7, and 9 are key features since they appear in each complex. To show the significance of a feature, a weight is associated with it. The feature with the biggest weight plays the most important role in determining the compliance of an example with a decision rule. The truth value of a decision rule is the sum of the weights of the satisfied conditions.

The weights of feature tests are calculated from the feature's frequencies and then normalized so that the sum of the weights in a decision rule is 1. For example, from the first solution of (14), the frequencies of the features F2, F7, F9 are 1.0 (4/4), and the frequencies of the features F1 and F_{10} are 0.5 (2/4). If a feature is assigned to take on values from the positive examples, a feature test is built to include all the values from the covered positive examples. If a feature is assigned to take on the values from negative examples, the feature test then uses operator $\neq$ and includes all the values from negative examples.

After the weights are normalized and all selectors are assigned values from positive examples, the decision rule is generated in the following format:

If $< F_2 = 2, 4, 5 >$ (0.25) and $< F_3 = 3, 5 >$ (0.125) and $< F_7 = 3, 4, 9 >$ (0.25) and $< F_9 = 2, 3, 9 >$ (0.25) and $< F_{10} = 2, 3, 5 >$ (0.125)

then the example belongs to Class #1.

Since each of the solutions in (14) was obtained with the minimum number of features, CLILP2 is designed to allow for arbitrary combination of rules. The combined rules are equally effective in distinguishing the positive examples from all the negative examples. The combined rules are likely to cover a wider range of the meaning a concept represents. By combining four solutions from (14), the new rule is stated as:

If $< F_1 = 1, 4, 5 >$ (0.1) and $<F_2 = 2, 4, 5 >$ (0.2) and $< F_3 = 3, 5 >$ (0.1) and $< F_9 = 2, 3, 5 >$ (0.1) and $< F_7 = 3, 4, 9 >$ (0.2) and $< F_9 = 2, 3, 9 >$ (0.2) and$< F_{10} = 2, 3, 5 >$ (0.1)

then the example belongs to Class #1.

Editor's note

Part II – "Experimental results and conclusions" will be published in the next issue of *Kybernetes* (Vol. 24 No. 3, 1995).

References

1. Quinlan, J.R., "Learning effective classification procedures and their application to chess ending games", in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, San Mateo, CA, 1983.
2. Cestnik, B., Konoenko, I. and Bratko, I., "ASSISTANT 86: a knowledge elicitation tool for sophisticated users", in Bratko, I. and Lavrac, N. (Eds), *Proceedings of the Second European Working Session on Learning*, Bled, Yugoslavia, May 1987.
3. Quinlan, J.R., "Probabilistic decision trees", in Kodratoff, Y. and Michalski, R.S. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann, San Mateo, CA, 1990, pp. 140-52.
4. Quinlan, J.R., "Generating production rules from decision trees", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, Morgan Kaufmann, San Mateo, CA, pp. 304-7.
5. Michalski, R.S., "On the quasi-minimal solution of the general covering problem", *Proceedings of the 5th International Symposium on Information Processing (FCIP 69)*, Vol. A3 (switching circuits), Bled, Yugoslavia, 1969, pp. 125-8.
6. Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., "The multipurpose incremental learning system AQ15 and its testing application to three medical domains", *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, Morgan Kaufmann, San Mateo, CA, pp. 1041-5.

7. Michalski, R.S., "Learning flexible concepts: fundamental ideas and a method based on two-tiered representation", in Kodratoff, Y. and Michalski, R.S. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann, San Mateo, CA, 1990, pp. 63-102.
8. Clark, P. and Niblett, T., "The CN2 algorithm", *Machine Learning*, Vol. 3, 1989, pp. 261-83.
9. Michalski, R.S. and Larson, J., "Selection of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11", *Reports of the Department of Computer Science*, No. 867, University of Illinois, Urbana, IL, 1978.
10. Quinlan, J.R., "The effect of noise on concept learning", in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. II, Morgan Kaufmann, San Mateo, CA, 1986.
11. Quinlan, J.R., "Simplifying decision trees", *International Journal of Man-Machine Studies*, Vol. 27, 1987, pp. 221-334.
12. Niblett, T., "Constructing decision trees in noisy domains", *Proceedings of the Second European Working Session on Learning*, Bled, Yugoslavia, 1987, Sigma Press, pp. 67-78.
13. Cios, K.J. and Liu, N., "Machine learning in generation of a neural network architecture: a continuous ID3 approach", *IEEE Transactions on Neural Networks*, Vol. 3 No. 2, 1992, pp. 280-91.
14. Michalski, R.S., "Variable-valued logic: system VL1", *Proceedings of the 1974 International Symposium on Multiple-valued logic on Pattern Recognition*, West Virginia University, Morgantown, VA, 1974, pp. 323-46.
15. Bar Yehuda, R. and Even, S., "A linear-time approximation algorithm for the weighted vertex cover problem", *Journal of Algorithms*, Vol. 2, 1981, pp. 198-203.
16. Cios, K.J. and Torkzadeh, G., "Fuzzy clustering and neural networks for instrument validation", *Heuristics, the Journal of Knowledge Engineering*, Vol. 5 No. 1, 1992, pp. 90-6.
17. Grafinked, R.S. and Nembauser, G.L., *Integer Programming*, John Wiley & Sons, New York, NY, 1972.
18. Bellmore, M. and Ratliff, H.D., "Set covering and involutory bases", *Management Science*, Vol. 18 No. 3, November 1971.
19. Chvatal, V., "A greedy-heuristic for the set-covering problem", *Mathematical Operations Research*, Vol. 4 No. 3, 1979.
20. Hochbaum, D.S., "Approximation algorithm for the weighted set-covering and vertex cover problems", *Siam Journal of Computing*, Vol. 11 No. 3, August 1982.
21. Aspvall, B. and Stone, R.E., "Khachiyan's linear programming algorithm", *Journal of Algorithms*, Vol. 1 No. 1, 1980, pp. 1-13.

Appendix

The special class of integer-programming problems, generally referred to as weighted set-covering problems, is of the form (for inequality):

$$\text{minimize} \quad Z = \sum_{j=1}^{n} w_j x_j$$

$$\text{subject to:} \quad \sum_{j=1}^{n} S_j x_j \geq b$$

and of the form (for equality):

$$\text{minimize} \quad Z = \sum_{j=1}^{n} w_j x_j$$

$$\text{subject to:} \quad \sum_{j=1}^{n} S_j x_j = b$$

where each $S_j = (s_{1j}, s_{2j}, \ldots, s_{mj})^t$ has $s_{ij} = 0$ or 1, $b = (1, 1, \ldots, 1)^t$, and w_j is a positive integer for $j = 1, 2, 3, \ldots, n$.

Since the set-covering problem is NP-complete, it is natural to search for polynomial-time approximation algorithms. The best known result was obtained by Chvatal[19]. His algorithm takes time $O(m \log n)$, where $m = \sum_{j=1}^{n} |S_j|$ and the solution of the objective function Z satisfies.

$$Z \leq Z_{\text{opt}} O(\log d)$$

where Z_{opt} is an optimal solution of objective function, and $d = \max_j |S_j|$.

Another polynomial-time approximation algorithm has been proposed by Hochbaum[20]. She uses a solution of a linear programming problem, and therefore the proof that her algorithm is polynomial relies on the Russian algorithm[21] for solving linear programming in polynomial time. The solution of the objective function of Hochbaum's algorithm satisfies the condition:

$$Z \leq Z_{\text{opt}} \cdot f$$

where f is the largest number of sets which contain a certain element. Let $e_1 e_2, \ldots, e_t$ be the element in U, and define

$$F(j) = \{i \mid e_j \in S_i\}$$

then $f = \max |F(j)|$.

The algorithm used in the CLILP2 machine learning algorithm was proposed by Bar Yehuda and Even[15]. Their algorithm bypasses the need to use a solution of linear programming and has running time linear to $\sum_{j=1}^{n} |S_j|$.

The algorithm uses the following notation. The input is a family of sets $S_1, S_2, \ldots, S_n$ and $N = \{1, 2, \ldots, n\}$. The universal set is $e_1, e_2, \ldots, e_t$, and $T = \{1, 2, \ldots, t\}$. For each $j \in S$, $F(j) = \{i \mid e_j \in S_i\}$. The given weight of S_i is w_i, while $SW(i)$ denotes the "residual" weight of S_i. For $j \in T$, $EW(j)$ the "weight" is assigned to e_i. I denotes the set of indices of sets already selected to be in a cover. J denotes the set of indices of elements not yet covered.

The algorithm follows:

```
(1) $Vi\ SW(i) \leftarrow \omega_i$, $[Vj\ EW(j) \leftarrow 0]$
(2) $I \leftarrow \varnothing, j \leftarrow T$.
(3) Do while $j \neq \varnothing$
(4) Let $j \in J$
(5) $M \leftarrow \text{Min}\{SW(i) \mid i \in F(j)\}, [EW(j) \leftarrow M]$
(6) Let $k$ ÂF(J) such that $SW(k) = M$.
(7) $Vi \in F(j) SW(i) \leftarrow SW(i) - M$.
(8) $I \leftarrow I \cup \{k\}, J \leftarrow J - S_k$
(9) end.
```

It has been proved by Bar Yehuda and Even[15] that the solution of objective function satisfies:

$$Z \leq Z_{\text{opt}} \underset{j \in T}{\text{Max}} |F(i) \cap I|.$$