Rule Induction with Extension Matrices

Xindong Wu
Department of Software Development
Monash University
900 Dandenong Road
Melbourne, VIC 3145, Australia

Journal of the American Society for Information Science
Volume 49, Number 5, 1998, pp 435–454

Abstract
This paper presents a heuristic, attribute-based, noise-tolerant data mining program, HCV (Version 2.0), based on the newly-developed extension matrix approach. By dividing the positive examples (PE) of a specific class in a given example set into intersecting groups and adopting a set of strategies to find a heuristic conjunctive formula in each group which covers all the group’s positive examples and none of the negative examples (NE), the HCV induction algorithm adopted in the HCV (Version 2.0) software finds a description formula in the form of variable-valued logic for PE against NE in low-order polynomial time at induction time. In addition to the HCV induction algorithm, this paper also outlines some of the techniques for noise handling and discretization of numerical domains developed and implemented in the HCV (Version 2.0) software, and provides a performance comparison of HCV (Version 2.0) with other data mining algorithms ID3, C4.5, C4.5rules and NewID in noisy and continuous domains. The empirical comparison shows that the rules generated by HCV (Version 2.0) are more compact than the decision trees or rules produced by ID3-like algorithms, and HCV's predicative accuracy is competitive with ID3-like algorithms.

1 Introduction
Artificial intelligence (AI) concerns itself with the problem of making machines perform tasks like vision, planning, and diagnosis, that usually require human intelligence. Machine learning research in AI is concerned with the problem of how machines can automatically acquire the knowledge that may enable them to perform those tasks. Along with the recognition of the so-called knowledge bottleneck problem (Feigenbaum, 1981) in transforming knowledge from human experts to knowledge-based systems, machine learning research has been expanding rapidly in recent years.

Research on machine learning has concentrated in the main on inducing rules from unordered sets of examples, especially attribute-based induction, an inductive formalism where examples are described in terms of a fixed collection of attributes. Learning from examples, or knowledge discovery in databases (KDD), has been seen (Michie, 1987; Quinlan, 1988; Wu, 1995) as not only a feasible way but also the only way of avoiding the knowledge bottleneck problem. While it is often difficult for an expert to articulate their expertise explicitly and clearly, it is usually relatively easy to document case studies of their skill at work. The learning systems in commercial use today are almost exclusively inductive ones. The most widespread family of learning algorithms for learning systems is the decision tree based ID3-like family (Hunt, Marin, & Stone, 1966; Quinlan, 1979; Cestnik, Kononenko, & Bratko, 1987; Quinlan, Compton, & Lazarus, 1987;
Utgoff, 1989; Quinlan, 1993; Zheng, 1995), which is of the attribute-based paradigm. However, a new family of inductive algorithms based on the extension matrix approach (Hong, 1985; Hong, & Uhrik, 1987; Hong, 1989a; Wu, 1992a; Wu, 1993a) has been recently proposed. This paper takes the newly-developed extension matrix approach, improves it to be competitive with ID3-like algorithms and does an empirical study of its properties.

In addition to interest from the knowledge-based systems community, KDD is itself a research frontier (Wu, 1993b) for both database technology and machine learning techniques, and has seen sustained research over recent years (Fayyad & Uthurusamy, 1995; Fayyad et al., 1996). It acts as a link between the two fields, thus offering a dual benefit. Firstly, since database technology has already found wide application in many fields, machine learning research obviously stands to gain from this greater exposure and established technological foundation. Secondly, as databases grow in both number and size, the prospect of mining them for new, useful knowledge, becomes yet more enticing. Machine learning techniques can augment the ability of existing database management systems to represent, acquire, and process a collection of expertise such as those which form part of the semantics of many advanced applications.

Generally speaking, all kinds of attribute-based induction algorithms can be adapted to extract knowledge from databases. It is not difficult to add an induction engine to an existing database system in an ad hoc manner to perform data mining, or design some specific engines to learn from domain-specific data sets. However, when we integrate machine learning techniques into database systems to build practical data mining systems, we must face a number of issues:

- Efficient induction algorithms are needed. The algorithms should be capable of being applied to realistic databases, e.g. $\geq 10^6$ relational tuples. Exponential or even medium-order polynomial complexity will not be of practical use.

- The knowledge acquired needs to be tested and/or used in the data mining systems.

- Noise (including missing information) has to be effectively handled. Data mining is different from mathematical induction. We cannot assume that the data in the given databases is complete. There are various sources of noise including missing values in real-world databases. To produce acceptable results for realistic applications, noise handling facilities are often essential in induction algorithms (Kibler & Aha, 1989; Schlimmer & Granger, 1989; Weiss, Galen, & Tadepalli, 1990; Wu, Krisár, & Málén, 1995).

- Numerical data and symbolic data are equally important in practical applications. Existing data mining algorithms can generally be divided into two groups: numerical methods, including statistical methods and neural networks, which are good at processing numerical data in noisy environments, and symbolic AI methods which are more efficient in dealing with symbolic or nominal data. It has been a long term dispute that AI methods (especially decision trees) are too simple to represent the real world. In the meanwhile, we can also easily argue that numerical methods are not good enough to represent and manipulate logic relationships among symbolic values. We need induction algorithms that can effectively deal with both types of data.

The ID3-like algorithms (such as ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993)) and HCV (Wu, 1993a) are all low-order polynomial in both time and space. However, since induction from databases relies to a great extent on the quality of the training data, noise handling and dealing with both numerical data and symbolic data are as important as the induction itself in realistic data mining applications.
The outline of this paper is as follows. In the following section, we give a simple example of attribute-based induction to show the difference between the rules in variable-valued logic produced by HCV, the decision tree generated by C4.5 (the most recent successor of ID3-like algorithms), and the decision tree’s compiled rules by C4.5rules. The extension matrix approach for data mining is outlined in Section 3. In Section 4 we describe the HCV algorithm in detail. In Sections 5 and 6 we outline briefly some of the techniques developed and implemented in the HCV (Version 2.0) program for noise handling and discretization of continuous domains respectively, and we follow these with a performance comparison of HCV with some famous ID3-like algorithms including C4.5 and C4.5rules on a collection of standard databases including the famous MONK’s problems in Section 7.

2 An Example of Attribute-based Induction

Given a discrete finite attribute space of $a$ dimensions, $E = D_1 \times D_2 \times \cdots \times D_a$, where each $D_j$ ($j = 1, \ldots, a$) is a finite set of symbolic values or a numerical domain, an example, or a case, $e = (V_1, \cdots, V_a)$ in $E$ means $V_j \in D_j$. A positive example is such an example that belongs to a known class which, say, has a specific name in $E$. All the other examples which do not belong to the known class can be called negative examples (NE) at the moment we are considering the known class. The induction task is to generate a description, say production rules or a decision tree, that covers all of the positive examples (PE) against¹ NE or classifies them correctly².

<table>
<thead>
<tr>
<th>ORDER</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>b</td>
<td>c</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>b</td>
<td>b</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>c</td>
<td>b</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>c</td>
<td>b</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>c</td>
<td>b</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>c</td>
<td>c</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>c</td>
<td>a</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>T</td>
</tr>
</tbody>
</table>

Example 1. Given 18 cases of $T$ and $F$ in Table 1. Each case is described by four attributes: $X_1 \{0,1\}$, $X_2 \{a, b, c\}$, $X_3 \{a, b, c\}$, and $X_4 \{0, 1\}$.

Running C4.5 (Quinlan, 1993)³ we get a decision tree in Figure 1 which is converted by C4.5rules in the C4.5 package into the conjunctive decision rules below:

¹against is used to mean that the description should cover none of the negative examples if PE and NE contain no common examples.
²We leave noise handling to Sections 5 and 6. For Sections 2 to 4, we assume no contradictory examples in the training set which have the same attribute values but different classifications, and require induction results to be consistent with the training examples.
³Since C4.5 is the most recent successor of ID3 and is commonly recognized as a state-of-the-art method for inducing decision trees (Dougherty, Kohavi, & Sahami, 1995) we compare our work on HCV mainly with C4.5 and its corresponding rule decompiler, C4.5rules, with only passing references to other algorithms such as ID3 and NewID.
Figure 1: A decision tree (by C4.5) for Example 1
Figure 2: Another decision tree (by C4.5) for Example 1

\[
\begin{align*}
X2 &= b \\
\lor \quad &X1 = 0 \land X3 = a & X1 = 1 \land X2 = a \\
\lor \quad &X1 = 0 \land X3 = b & X1 = 1 \land X2 = c \\
\lor \quad &X1 = 0 \land X2 = a & X2 = c \land X3 = c \\
\rightarrow \quad &\text{The } T \text{ class.} & \rightarrow \quad &\text{The } F \text{ class.}
\end{align*}
\]

Running C4.5 with the \texttt{-s} option, which causes the values of discrete attributes to be grouped rather than having a separate branch for every possible value of the attribute, we get the decision tree in Figure 2. The tree is converted by \texttt{C4.5rules} into the conjunctive decision rules below:

\[
\begin{align*}
X2 &= [b] \\
\lor \quad &X1 = 0 \land X2 = [a] & X2 = [c] \land X3 = [c] \\
\lor \quad &X1 = 0 \land X3 = [a,b] & X1 = 1 \land X2 = [a,c] \\
\rightarrow \quad &\text{The } T \text{ class.} & \rightarrow \quad &\text{The } F \text{ class.}
\end{align*}
\]

Running the HCV algorithm in Section 4, we can get the following rule in variable-valued logic (Michalski, 1975) for the \texttt{T} class against examples of the \texttt{F} class:

\[
\begin{align*}
&\{ X2 = [b] \} \\
\lor \quad &\{ X1 = [0] \} \\
&\quad \{ X2 = [a] \} \\
\lor \quad &\{ X1 = [0] \} \\
&\quad \{ X4 = [0] \} \\
\rightarrow \quad &\text{The } T \text{ class.}
\end{align*}
\]

As we can easily see, the results produced by each of \texttt{C4.5}, \texttt{C4.5rules} and \texttt{HCV} are more compact and more general than the original cases in Table 1. They take less space and can be used to predict or classify new examples.
In contrast to credit assignment and generate-and-test processes in genetic algorithms (Carbonell, 1990) and numerical activity vectors based numerical computations in connectionist methods (Dayhoff, 1990), attribute-based induction concentrates on symbolic and heuristic computations. These relate to models that operate at the level of symbols and operations that manipulate symbolic expressions with an emphasis on heuristic rather than computationally explosive optimization strategies. No explicit credit assignment strategies are necessary in the attribute-based induction paradigm.

One of the great advantages of the attribute-based paradigm is the fact that it does not require users to specify explicit, built-in, background knowledge in the form of, say, generalization hierarchies, although there is some implicit background knowledge embedded in the representation of examples. This means that attribute-based induction algorithms can be applied to any syntactically well-formed databases. This feature of the attribute-based paradigm differs greatly from explanation-based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) and inductive logic programming (Muggleton, 1992).

3 The Extension Matrix Approach

The new family of inductive algorithms based on the extension matrix approach was first developed at the University of Illinois by Hong et al. (Hong, 1985; Hong, & Uhrik, 1987) and then redesigned by the author (Wu, 1992a; Wu, 1993a). In contrast to decision trees in ID3-like algorithms, the algorithms of the extension matrix based family take a new kind of matrix, called an extension matrix, as their mathematical basis.

3.1 Terminology and notation

Let $a$ be the number of attributes $\{X_1, \ldots, X_a\}$ in an example space, $n$ be the number of negative examples in a training set, and $p$ be the number of positive examples. Let $NE$ be expressed by

$$NEM = (e_1, \ldots, e_n)^T = (r_{ij})_{n \times a}$$

(1)

with the $i$-th negative example $e_i^-(i = 1, \ldots, n)$ being expressed on the $i$-th row of matrix $NEM$ and $NEM(i, j) = r_{ij}$ indicating that the value of $e_i^-$ on attribute $X_j$ is $r_{ij}$. $(e_1^-, \ldots, e_n^-)^T$ denotes the transpose of matrix $(e_1^-, \ldots, e_n^-)$.

**Definition 1.** Let the $k$-th ($k = 1, \ldots, p$) positive example be expressed as $e_k^+ = (v_{ik}^+, \ldots, v_{ak}^+)$. The matrix below is the extension matrix of $e_k^+$ against $NE$

$$EM_k = (r_{ij})_{n \times a}$$

(2)

where

$$r_{ij} = \begin{cases} * & \text{when } v_{ij}^+ = NEM_{ij} \\ NEM_{ij} & \text{when } v_{ij}^+ \neq NEM_{ij} \end{cases}$$

and ‘*’ denotes a dead element which cannot be used to distinguish the positive example from negative examples.

**Definition 2.** In an $EM_k$, a set of $n$ nondead elements $r_{ij}$ ($i = 1, \ldots, n$, $j_i \in \{1, \ldots, a\}$) that come from the $n$ different $i$ rows is called a path in the extension matrix. For example, $\{X_1 = 1, X_2 = 0, X_1 = 1\}$ and $\{X_1 = 1, X_3 = 1, X_2 = 0\}$ are two different paths in the extension matrix below.

---

4Núñez argued that for this reason, most of the time the ID3 family of algorithms are neither logical nor understandable to experts and he made some improvements (i.e., executing different types of generalization and reducing the classification cost) on ID3 in his algorithm by means of background knowledge in (Núñez, 1991).
\[
\begin{pmatrix}
X_1 & X_2 & X_3 \\
1 & * & * \\
* & 0 & 1 \\
1 & 0 & *
\end{pmatrix}
\]

Lemma (Hong, 1989a). A path \( \{ r_{ij_1}, \ldots, r_{ijn} \} \) in an \( EM_k \) corresponds to a conjunctive formula

\[
L = \bigwedge_{i=1}^{n} [X_{ji} \neq r_{ij}]
\]

which covers \( e^+_k \) against \( NE \) and vice versa.

Each \( [X_{ji} \neq r_{ij}] \) here is a selector in variable-valued logic (Michalski, 1975). If \( r_{ij} \) appears on \( m \) \((m \in \{0, \ldots, n\})\) rows in the same column \( j_i \) of an \( EM_k \), we say it or \( [X_{ji} \neq r_{ij}] \) covers \( m \) rows of the \( EM_k \).

Definition 3. Matrix \( EMD = (r_{ij})_{n \times a} \) with

\[
r_{ij} = \begin{cases} 
* & \text{when } \exists k_1 \in \{i_1, \ldots, i_k\} : EM_{k_1}(i, j) = \ast \\
\lor_{k=1}^a EM_{ik_2}(i, j) = NEM(i, j) & \text{otherwise}
\end{cases}
\]

is called the disjunction matrix of the positive example set \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \) or the disjunction matrix of \( EM_{i_1}, \ldots, EM_{i_k} \).

Definition 4. In the \( EMD \) of a positive example set \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \), a set of \( n \) nondead elements \( r_{ij} \) \((i = 1, \ldots, n, j_i \in \{1, \ldots, a\})\) that come from the \( n \) different \( i \) rows is also called a path.

Theorem 1. A path \( \{ r_{ij_1}, \ldots, r_{ijn} \} \) in the \( EMD \) of \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \) corresponds to a conjunctive formula or cover

\[
L = \bigwedge_{i=1}^{n} [X_{ji} \neq r_{ij}]
\]

which covers all of \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \) and vice versa.

Proof. If \( EMD(i,j) = \ast \), there must be a \( k_2 \in \{i_1, \ldots, i_k\} \) and \( EM_{k_2}(i, j) = \ast \), which means there is no common nondead element on the \((i,j)\)-position of all the extension matrices \( EM_{i_1}, \ldots, EM_{i_k} \). If there exists no dead element on the \((i,j)\)-position of any \( EM_{k_2}(k_2 \in \{i_1, \ldots, i_k\}) \), it is certain that \( EM_{k_2}(i, j) = NEM(i, j) \) according to Definition 1 and that the \( NEM(i,j) \) is a common nondead element in \( EM_{i_1}, \ldots, EM_{i_k} \) according to Definition 3. Therefore, the common nondead elements in all the extension matrices \( EM_{i_1}, \ldots, EM_{i_k} \) and the nondead elements in their disjunction matrix \( EMD \) correspond to each other and each common path in \( EM_{i_1}, \ldots, EM_{i_k} \), formed by the common nondead elements in every \( EM_{ik_2}(k_2 = 1, \ldots, k) \), corresponds to a path in \( EMD \) and vice versa. According to the above Lemma, the formula which corresponds to a path in \( EMD \) must be a common formula for all of the \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \).

If there is no path which covers all the \( n \) rows in \( EMD \), there is no common path and therefore no conjunctive formula in all the extension matrices \( EM_{i_1}, \ldots, EM_{i_k} \).

Definition 5. If there exists at least one path in the \( EMD \) of a positive example set \( \{ e^+_{i_1}, \ldots, e^+_{i_k} \} \) against \( NE \), all the positive examples in the set intersect and the positive example set is called an intersecting group.

Theorem 2. For a given set of examples, if PE and NE are persistent, which means they contain no common examples, there always exists at least one conjunctive formula covering any positive example \( e^+_k \in PE \) against \( NE \).
Proof. As PE and NE are persistent, we can always find at least one nondead element on each row $i$ in the $EM_k$ of $e_i^+$ against NE which discriminates $e_i^+$ and the $i$-th negative example $e_i^-$ in NEM. Therefore, we can always find at least one path which corresponds to a conjunctive formula cover in $EM_k$.

From Definition 5 and the proof process of Theorem 2, we can easily get the following corollary.

Corollary. For a given set of examples, if PE and NE are persistent, there always exists at least one conjunctive formula for each intersecting example group.

3.2 Optimization problems

There are two striking optimization problems in the extension matrix approach:

- The minimum formula (MFL) problem: Generating a conjunctive formula that covers a positive example or an intersecting group of positive examples against NE and has the minimum number of different conjunctive selectors.

- The minimum cover (MCV) problem: Seeking a cover which covers all positive examples in PE against NE and has the minimum number of conjunctive formulae with each conjunctive formula being as short as possible.

The extension matrix $EM_k$ of each positive example $e_i^+$ against NE contains all such paths that each correspond to a conjunctive formula of $e_i^+$ against NE. An optimal cover of PE against NE is such a minimum set of formulae that is a logical combination of all the formulae from every $EM_k$ ($k = 1, \ldots, p$). Therefore, both MFL and MCV problems have been proved to be NP-hard (Hong, 1985).

Two complete algorithms are designed to solve the optimization problems MFL and MCV in (Wu, 1992a). When each attribute domain $D_i (i = 1, \ldots, a)$ satisfies $|D_i| = 2$, the time complexity for these two algorithms is $O(na2^a)$ and $O(n2^{na} + pa2^a)$ respectively. When there exists $|D_j| > 2$ or $D_j$ is a real-valued interval ($j \in \{1, \ldots, a\}$), a decomposition method that decomposes $D_j$ into several sub-domains, whose bases are each two, is also designed in (Wu, 1992a).

3.3 Heuristic strategies in AE1

As the nature of the MFL and MCV problems is NP-hard, when an example set or an attribute space is large the induction process based on the complete algorithms will become computationally intractable. Two strategies are adopted in AE1 (Hong, 1985) to find approximate rather than optimal solutions for both MFL and MCV problems:

1. Starting search from the columns with the most nondead elements, and

2. Simplifying redundancy by deductive inference rules in mathematical logic.

There are two problems in AE1. First, its first strategy can easily lose optimal solutions in some cases. Taking the simple extension matrix below as an example, the first heuristic strategy in AE1 cannot produce the optimal formula as it will choose the selector $[X_2 \neq 0]$ first. Second, simplifying redundancy for MFL and MCV problems is NP-hard. No heuristic strategy for this process has been reported.
\[
\begin{pmatrix}
X_1 & X_2 & X_3 \\
1 & * & * \\
* & 0 & 1 \\
1 & 0 & * \\
* & 0 & 1 \\
1 & 0 & * \\
* & * & 1
\end{pmatrix}
\]

The HCV algorithm in the next section is a heuristic algorithm designed to solve the MCV problem with the two disadvantages of AE1 removed.

4 HCV: A Heuristic Covering Algorithm

Time complexity and description compactness are two important criteria for all induction algorithms. In the extension matrix approach, there are two extreme strategies which each place special emphasis on only one of the two criteria. The first is to initially find all possible formulae from each positive example’s extension matrix, and then take an exhaustive search among all the formulae to find the shortest combination which covers all positive examples. This strategy can give the shortest description in the form of variable-valued logic but works in exponential time. The second is simply separating one positive example from NE by memorizing the positive example or all positive examples in PE from NE by memorizing each of the positive examples into a boolean OR formula. This trivial heuristic can work quickly but generates an extremely large description. An OR formula of this kind cannot be used directly to classify new examples which have not been presented in the training example set, while simplifying it into the shortest form also needs NP-hard time. Therefore, a good induction algorithm should be able to either avoid the NP-hard time or produce a more brief description which is at least able to correctly classify the PE and NE in a given training example set if the training set is persistent (see Theorem 2). In this section, we will show that the HCV algorithm has made progress on both the time and the description compactness.

4.1 The HFL algorithm

The HFL algorithm is designed to find a heuristic conjunctive formula which corresponds to a path in an extension matrix or a disjunction matrix when there is at least one path in the disjunction matrix. As a disjunction matrix can be processed in the same way as an extension matrix to find its conjunctive formulae, we will refer only to the extension matrices below.

4.1.1 Four strategies in HFL

Four strategies are adopted in the HFL algorithm:

1. The fast strategy. In an extension matrix \( EM_k = (r_{ij})_{n \times a} \), if there is no dead element in a (say \( j \)) column, then \( [X_j \neq r_j] \) where \( r_j = \lor_{i=1}^n r_{ij} \) is chosen as the one selector cover for \( EM_k \).

For example, selector \( [X_5 \neq \{ \text{normal, dry – peep} \}] \) below can cover all the five rows in the extension matrix.

\[
\begin{pmatrix}
X_1 & X_2 & X_3 & X_4 & X_5 \\
\text{absent} & \text{slight} & \text{strip} & * & \text{normal} \\
* & * & \text{hole} & \text{fast} & \text{dry – peep} \\
\text{low} & \text{slight} & \text{strip} & * & \text{normal} \\
\text{absent} & \text{slight} & \text{spot} & \text{fast} & \text{dry – peep} \\
\text{low} & \text{medium} & * & \text{fast} & \text{normal}
\end{pmatrix}
\]
2. The \textit{precedence} strategy. When a $r_{ij}$ in column $j$ is the only nondead element of a row $i$ in an extension matrix $EM_k = (r_{ij})_{n \times a}$, the selector $[X_j \neq r_j]$ where $r_j = \vee_{i=1}^{n} r_{ij}$ is called an inevitable selector and thus is chosen with top precedence.

For example, $[X_1 \neq 1]$ and $[X_3 \neq 1]$ are two inevitable selectors in the extension matrix below which we have mentioned in Section 3.3.

\[
\begin{array}{ccc}
X_1 & X_2 & X_3 \\
1 & * & * \\
* & 0 & 1 \\
1 & 0 & * \\
* & 0 & 1 \\
1 & 0 & * \\
* & * & 1 \\
\end{array}
\]

3. The \textit{elimination} strategy. When each appearance of some nondead element in the $j_1$-th column of some row is always coupled with another nondead element in the $j_2$-th column of the same row in an extension matrix $EM_k = (r_{ij})_{n \times a}$, $[X_{j_1} \neq r_{j_1}]$, where $r_{j_1} = \vee_{i=1}^{n} r_{j_1}$, is called an eliminable selector and thus is eliminated by selector $[X_{j_2} \neq r_{j_2}]$ where $r_{j_2} = \vee_{i=1}^{n} r_{j_2}$.

For example, attribute $X_2$ can be eliminated by attribute $X_3$ below.

\[
\begin{array}{cccc}
X_1 & X_2 & X_3 & X_4 \\
1 & * & 1 & * \\
* & 0 & 1 & 0 \\
1 & 0 & 1 & * \\
* & 0 & 1 & * \\
1 & 0 & 1 & * \\
1 & * & * & 0 \\
\end{array}
\]

4. The \textit{least-frequency} strategy. When all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, exclude a least-frequency selector which has least nondead elements in its corresponding column in the extension matrix.

For example, attribute $X_1$ in the following extension matrix can be eliminated and there still exists a path.

\[
\begin{array}{ccc}
X_1 & X_2 & X_3 \\
1 & * & 1 \\
* & 0 & 1 \\
1 & 0 & * \\
* & 0 & 1 \\
1 & 0 & * \\
* & 0 & 1 \\
\end{array}
\]

\textbf{Theorem 3.} All of the fast, precedence and elimination strategies are complete, which means if there exists one or more shortest conjunctive formulae in an extension matrix, they will not lose it.

\textbf{Proof.} (a) For an extension matrix $EM_k = (r_{ij})_{n \times a}$, the largest and the possible least numbers of different selectors in a path are $n$ and $1$ respectively. When $n$ selectors, whether
the same or different, from \( n \) different rows but from the same column \( j \) form a path, they can be integrated into one, \([X_j \neq \bigvee_{i=1}^n r_{ij}]\). So the selector found by the fast strategy must be an optimal formula of the extension matrix.

(b) When \( r_{ij} \) in column \( j \) is the only nondead element of some row \( i \) in an extension matrix, the selector \([X_j \neq r_{ij}]\) needs to appear or to be integrated into a complex selector like \([X_j \neq \{\cdots, r_{ij}, \cdots\}]\) in any path of the extension matrix. Picking up such kinds of selectors with top precedence will not violate the correctness of any path which is built but will speed up the construction process.

(c) For any path \( \{\cdots, r_{j_1}, \cdots\} \) in an extension matrix, we can simply replace \( r_{j_1} \) with \( r_{j_2} \) and \( \{\cdots, r_{j_2}, \cdots\} \) is also a path in the same extension matrix when \( r_{j_2} = \bigvee_{i=1}^n r_{ij_2} \), and each appearance of some nondead element in the \( j_1 \)-th column of some row is always coupled with another nondead element in the \( j_2 \)-th column of the same row. If there is another \( r_{j_3} \) in the path whose corresponding selector \([X_j \neq r_{j_3}]\) is also eliminable by \([X_j \neq r_{j_2}]\), we can simplify the path by replacing \( r_{j_1} \) and \( r_{j_2} \) with \( r_{j_2} \). Therefore, the elimination strategy is complete for constructing optimal covers.

**Theorem 4.** If there exists at least one shorter path which has less than \( n \) different conjunctive selectors in an extension matrix, the solution generated by the least-frequency strategy must be the shorter one.

**Proof.** When all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, there must exist at least one redundant selector in the extension matrix. For example, suppose there are \( k \) \((k \leq n)\) rows in the extension matrix which have not been covered and all the nondead elements on those rows are \( r_{i_1j_1}, \cdots, r_{i_1j_{1_1}}, \cdots, r_{i_kj_{1_1}}, \cdots, r_{i_kj_{1_k}}, \) as no more inevitable selector can be found at this moment, each row must contain at least two nondead elements. Therefore, crossing out any column can guarantee that each of those rows will still contain at least one nondead element, which means there is still at least one path left in the extension matrix. We can thus obtain the correctness proof of the fourth strategy above. As different selectors from the same column in an extension matrix can be integrated into one, excluding a column means there are at most \( n - 1 \) selectors in the paths left. So the paths after this strategy has been applied must be shorter than the trivial ones.

Although the column with the least nondead elements is not necessarily removed from all the optimal paths, the removal looks reasonable as choosing a column with fewer nondead elements means more columns thus more selectors may be involved in connecting a path. So the fourth strategy is a sensible heuristic.

### 4.1.2 Algorithm description

The HFL algorithm applies the four strategies in turn on an extension matrix or disjunction matrix. After a selector is chosen by the precedence strategy, the selector is added to the output variable of HFL, \( Hfl \), and all the rows covered by the chosen selector are labeled as covered to avoid further consideration. The four strategies are run on the uncovered rows only afterward. Words between /\* and */ are explanatory notes.

**Procedure HFL(EM; Hfl)**

1. **S0:** \( Hfl \leftarrow \{\} \)
2. **S1:** /* the fast strategy */
   - Try the fast strategy on all these rows which haven’t been covered;
   - If successful, add a corresponding selector to \( Hfl \) and return(\( Hfl \)).
S2: /* the precedence strategy */
    Apply the precedence strategy to the uncovered rows;
    If some inevitable selectors are found,
        add them to Hfl, label all the rows they cover,
        and go to S1.
S3: /* the elimination strategy */
    Apply the elimination strategy to those attributes
        that have neither been selected nor eliminated;
    If an eliminable selector is found,
        reset all the elements in the corresponding
        column with *, and go to S2.
S4: /* the least-frequency strategy */
    Apply the least-frequency strategy to those attributes
        which have neither been selected nor eliminated,
        and find a least-frequency selector;
    Reset all the elements in the corresponding column
        with *, and go to S2.
Return(Hfl)

Steps S1, S2, S3, and S4 implement the fast, precedence, elimination and least-frequency strategies introduced in Section 4.1.1. Once the fast strategy finds a column that has nondead elements on all the uncovered rows in an EM, the EM can be fully covered and thus the Hfl is ready. After one or more inevitable selectors have been chosen in Step S2, HFL will come back to Step S1 to test the fast strategy on uncovered rows. After one or more columns have been crossed out by the elimination strategy in Step S3, the precedence strategy and the fast strategy will be tested again. Only in those cases when all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, the least-frequency strategy is used and it can always cross out a column that has not been crossed out before. After Step S4, HFL comes back to test the precedence strategy. Because excluding a column by either the elimination strategy in Step S3 or the least-frequency strategy in Step 4 does not cover any uncovered rows, the fast strategy cannot be applicable immediately after these two strategies. This is why the control in HFL comes back to Step 2 instead of Step 1 at the end of both Step 3 and Step 4. Each time the control comes back to Step S2 or Step S1, there is at least one column or selector that has been processed, either crossed or crossed out. There are a columns in an EM in total, therefore at most a loops in HFL are needed.

The time complexity for Steps S1, S2, S3, and S4 is $O(na)$, $O(na)$, $O(na^2)$ and $O(na)$, respectively. The time complexity for the whole algorithm is thus

$$O(a(na + na + na^2 + na)) \approx O(na^3).$$

When the EM in the HFL algorithm is the extension matrix $EM_k$ of a positive example $e_k^+ = (v_{1k}^+, \ldots, v_{nk}^+)$ against NE, a selector $[X_j \neq r_j]$ in the Hfl is equivalent to $[X_j = v_{jk}^+]$ with existing examples in a given example set. Meanwhile, if the EM is the disjunction matrix $EMD$ of an intersecting group of positive examples $e_{i1}^+, \ldots, e_{ik}^+$ against NE, a selector $[X_j \neq r_j]$ is equivalent to $[X_j \in \lor_{k=1}^{k} v_{j_{ik}}^+]$ in the context of existing examples. When $X_j$ is a numerical attribute Section 6 outlines techniques to discretize it into a number of intervals each of which is treated in the same way as a symbolic value. In implementation (see Section 4.4), HFL produces
Table 2: Cases of Pneumonia and Tuberculosis

<table>
<thead>
<tr>
<th>ORDER</th>
<th>FEVER</th>
<th>COUGH</th>
<th>X-RAY</th>
<th>ESR</th>
<th>AUSCULTATION</th>
<th>DISEASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>high</td>
<td>heavy</td>
<td>flack</td>
<td>normal</td>
<td>bubble-like</td>
<td>Pneumonia</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
<td>heavy</td>
<td>flack</td>
<td>normal</td>
<td>bubble-like</td>
<td>Tuberculosis</td>
</tr>
<tr>
<td>3</td>
<td>low</td>
<td>slight</td>
<td>spot</td>
<td>normal</td>
<td>dry-peep</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>high</td>
<td>medium</td>
<td>flack</td>
<td>normal</td>
<td>bubble-like</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>medium</td>
<td>slight</td>
<td>flack</td>
<td>normal</td>
<td>bubble-like</td>
<td></td>
</tr>
</tbody>
</table>

conjunctions in the form of both \([X_j \neq r_j]\) and \([X_j \in \vee_{k=1}^{k_2} v_{h_{k_2}}^+]\) and chooses whichever is the shorter.

From Theorems 3 and 4, we can easily get the theorem below (Theorem 5).

**Theorem 5.** If there exists at least one path in an EM, the HFL algorithm can produce a conjunctive formula which corresponds to a path in the EM. The number of selectors in \(Hfl\) produced by HFL is always smaller than \(n\) so long as there is at least one path with less than \(n\) different elements in the EM.

### 4.1.3 An example run of HFL on Table 2

**Example 2** (from (Hong, 1989b)). Given sets PE and NE of cases of Pneumonia and Tuberculosis in Table 2. There are 5 positive examples for Pneumonia. \(PE = \{e_1^+, e_2^+, e_3^+, e_4^+, e_5^+\}\), \(NE = \{e_1^-, e_2^-, e_3^-, e_4^-, e_5^-\}\), and

\[
NEM = \begin{pmatrix}
\text{absent} & \text{slight} & \text{strip} & \text{normal} & \text{normal} \\
\text{high} & \text{heavy} & \text{hole} & \text{fast} & \text{dry-peep} \\
\text{low} & \text{slight} & \text{strip} & \text{normal} & \text{normal} \\
\text{absent} & \text{slight} & \text{spot} & \text{fast} & \text{dry-peep} \\
\text{low} & \text{medium} & \text{flack} & \text{fast} & \text{normal} \\
\end{pmatrix}
\]

The disjunction matrix \(EMD\) of \(\{e_1^+, e_2^+, e_3^+, e_4^+, e_5^+\}\) against \(NE\) is

\[
EMD = \begin{pmatrix}
\text{absent} & \ast & \text{strip} & \ast & \text{normal} \\
\ast & \ast & \text{hole} & \text{fast} & \ast \\
\ast & \ast & \text{strip} & \ast & \text{normal} \\
\ast & \ast & \text{fast} & \ast & \ast \\
\ast & \ast & \ast & \text{fast} & \text{normal} \\
\end{pmatrix}
\]

by Definition 3.

Running HFL on \(EMD\) eliminates attribute \(FEVER\) by the least-frequency strategy during the first loop of Steps S1, S2, S3 and S4 and \(EMD\) becomes

\[
\begin{pmatrix}
\ast & \ast & \text{strip} & \ast & \text{normal} \\
\ast & \ast & \text{hole} & \text{fast} & \ast \\
\ast & \ast & \text{strip} & \ast & \text{normal} \\
\ast & \ast & \ast & \text{fast} & \ast \\
\ast & \ast & \ast & \ast & \text{fast} & \text{normal} \\
\end{pmatrix}
\]

In the second loop, \([ESR \neq \text{fast}]\) which is equivalent to \([ESR = \text{normal}]\) is chosen as an inevitable selector on the fourth row by the precedence strategy and it covers rows 2, 4 and 5. \(EMD\) now becomes
Figure 3: A decision tree (by ID3 and C4.5) for Example 2

\[
\begin{align*}
(* & * \text{ strip} & * \text{ normal}) \\
(* & * & * & * & *)
\end{align*}
\]

In the third loop, attribute \( X - \text{ray} \) is eliminated by attribute \( \text{AUSCULTATION} \) by the elimination strategy and \( \text{EMD} \) becomes

\[
\begin{align*}
(* & * & * & * & * & \text{normal}) \\
(* & * & * & * & *)
\end{align*}
\]

Selector \([\text{AUSCULTATION} \neq \text{normal}]\) is finally chosen by the fast strategy to cover the remaining rows 1 and 3.

Therefore, the \( Hfl \) for \( \text{EMD} \) generated by the HFL algorithm is

\[ [\text{ESR} = \text{normal}] \land [\text{AUSCULTATION} \neq \text{normal}] \]

which covers all the five examples of \( \text{Pneumonia} \) against examples of \( \text{Tuberculosis} \).

Meanwhile, the decision trees generated by ID3 and C4.5 on the example set in Table 2 are the same as given in Figure 3, which is converted by C4.5rules into the rules below:

if \( \text{AUSCULTATION} = \text{bubble-like} \) then \( \text{Pneumonia} \);
if \( \text{AUSCULTATION} = \text{dry-peep} \) \& \( \text{ESR} = \text{normal} \) then \( \text{Pneumonia} \)
if \( \text{ESR} = \text{fast} \) then \( \text{Tuberculosis} \); and
if \( \text{AUSCULTATION} = \text{normal} \) then \( \text{Tuberculosis} \).
4.2 The HCV Algorithm

4.2.1 Algorithm description

The basic idea for the HCV algorithm is to first partition $PE$ of a specific class into $p'$ ($p' \leq p$) intersecting groups; call the heuristic Algorithm HFL to find a $Hfl$ for each intersecting group; then give the covering formula by logically ORing all the $Hfl$'s finally.

Procedure \texttt{HCV}(EM$_1$, $\ldots$, EM$_p$; Hcv)

integer n, a, p
matrix EM$_1$(n,a), $\ldots$, EM$_p$(n,a), D(p)
set Hcv
S1: \( d \leftarrow 0 \) /* \( D(j) = 1 (j = 1, \ldots, p) \) indicates that \( EM_j \) has been put into an intersecting group. */
Hcv$\leftarrow$$\emptyset$ /* initialization */
S2: for \( i = 1 \) to \( p \) do
\hspace{1cm} \text{if} \ D(i) = 0 \ \text{then}
\hspace{2cm} \{ \ \text{EM$\leftarrow$EM$_i$}
\hspace{3cm} \text{for} \ j = i+1 \ \text{to} \ p \ \text{do}
\hspace{4cm} \text{if} \ D(j) = 0 \ \text{then}
\hspace{5cm} \{ \ \text{EM$_2$} \leftarrow \text{EM} \cup \text{EM$_j$}
\hspace{6cm} \text{If there exists at least}
\hspace{7cm} \text{one path in EM$_2$}
\hspace{8cm} \text{then} \ \{ \ \text{EM$\leftarrow$EM$_2$}, \ D(j)$\leftarrow$1 \} 
\hspace{5cm} \}
\hspace{3cm} \}
\hspace{2cm} \text{next} \ j
\hspace{1cm} \text{call} \ \text{HFL}(\text{EM}; \text{Hfl})
\hspace{1cm} \text{Hcv$\leftarrow$Hcv$\lor$Hfl}
\hspace{1cm} \}

\text{next} \ i

Return(Hcv)

Step S1 in the algorithm requires \( p + 1 \) operations. The time complexity for implementing \( \text{EM$_2$} \leftarrow \text{EM} \cup \text{EM$_j$} \) by Definition 3 is \( O(na) \) and for testing if there is at least one path in \( \text{EM$_2$} \) is also \( O(na) \). The worst case operation for Step S2 is

\[
O\left( \sum_{i=1}^{p} (na + \sum_{j=i+1}^{p} (2na + na + na + 1) + (na^3) + 1) \right)
\]

\[
\approx O(pna^3 + p^2na).
\]

So the time complexity for Algorithm HCV is \( O(pna^3 + p^2na) \).

**Theorem 6.** The formula \( Hcv \) generated by the HCV algorithm covers all the positive examples against negative examples in a given example set if positive examples and negative examples are persistent.

**Proof.** Each \( Hfl \) in the \( Hcv \) formula produced by the HCV algorithm covers a group of positive examples against \( NE \). So no negative example in \( NE \) will be covered by any \( Hfl \). Neither will the \( Hcv \) cover any of the negative examples in \( NE \) (because it is an OR combination of all the \( Hfls \)). As all positive examples have been included in the intersecting groups in Step S2, each positive example is covered by a \( Hfl \) in the \( Hcv \).

Algorithm HCV is a bidirectional algorithm. It first groups the positive example set in a top-down manner and then calls algorithm HFL, which works using a bottom-up approach. Its time complexity is low-order polynomial as opposed to exponential in the first strategy mentioned.
Table 3: Cases of Play and Don’t Play (adapted from (Quinlan, 1986))

<table>
<thead>
<tr>
<th>ORDER</th>
<th>OUTLOOK</th>
<th>TEMPERATURE</th>
<th>HUMIDITY</th>
<th>WINDY</th>
<th>DECISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rain</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>2</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>4</td>
<td>overcast</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>6</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>7</td>
<td>sunny</td>
<td>hot</td>
<td>normal</td>
<td>true</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>11</td>
<td>rain</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>12</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>13</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>Don’t Play</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>Don’t Play</td>
</tr>
</tbody>
</table>

at the beginning of Section 4. From Theorems 3 and 4 and the Corollary in Section 3.1, both Algorithm HFL and Algorithm HCV usually produce shorter formulae than the trivial strategy (the second strategy), also mentioned at the beginning of Section 4, so long as the shorter formulae exist.

**Theorem 7.** If there exists at least one conjunctive cover in a given training example set, the formula produced by HCV must be conjunctive.

**Proof.** If there exists at least one conjunctive cover in a given training example set, there must exist at least one path in the disjunction matrix of all the positive examples against the negative examples in the given example set according to Theorem 1. Therefore, all the positive examples will be put into an intersecting group in Step 2 of HCV and a conjunctive \( Hfl \) will be produced by calling HFL as the solution.

Example 2 in Section 4.1.3 illustrates the theorem.

### 4.2.2 Two example runs of HCV

**Example 3.** Table 3 shows a set of training examples for deciding whether to play golf on a Saturday afternoon.

Considering PE (of Play) and NE (of Don’t Play) in Table 3, let us observe the results generated by the HCV algorithm.

For the given example set, \( NE = \{ f_1, e_2, e_7, e_8, e_1, e_9, e_{10}, e_{11} \} \) and \( PE = \{ e_3^+, e_4^+, e_5^+, e_6^+, e_9^+ \} \)

\[
NEM = \begin{pmatrix}
\text{rain} & \text{hot} & \text{high} & \text{true} \\
\text{rain} & \text{cool} & \text{normal} & \text{true} \\
\text{sunny} & \text{hot} & \text{normal} & \text{true} \\
\text{sunny} & \text{mild} & \text{high} & \text{true} \\
\text{sunny} & \text{hot} & \text{high} & \text{false} \\
\text{sunny} & \text{cool} & \text{normal} & \text{false} \\
\text{rain} & \text{mild} & \text{normal} & \text{true} \\
\end{pmatrix}
\]

The first intersecting group found in Step S2 by starting with the first positive example \( (e_3^+) \) and calling the GEM and IDEN algorithms is \( \{ e_3^+, e_4^+, e_6^+ \} \) and the disjunction matrix \( EMD \).
against NE is

\[
EMD_1 = \begin{pmatrix}
  \text{rain} & \text{hot} & * & * \\
  \text{rain} & * & * & * \\
  \text{sunny} & \text{hot} & * & * \\
  \text{sunny} & * & * & * \\
  \text{sunny} & * & * & *
\end{pmatrix}.
\]

Calling HFL, \([O\text{UTLOOK} \neq \{\text{rain, sunny}\}]\) which is equivalent to \([O\text{UTLOOK} = \text{overcast}]\) is chosen\(^5\) by the fast strategy and the first \(Hfl\) is thus

\([O\text{UTLOOK} = \text{overcast}]\).

The second intersecting group found in Step S2 by starting with the third positive example \((e_3^+)\) and calling the GEM and IDEN algorithms is \(\{e_3^+, e_{10}^+, e_{11}^+\}\) and the disjunction matrix \(EMD_2\) is

\[
EMD_2 = \begin{pmatrix}
  * & * & * & \text{true} \\
  * & * & * & \text{true} \\
  \text{sunny} & * & * & \text{true} \\
  \text{sunny} & * & * & * \\
  \text{sunny} & * & * & * \\
  * & \text{mild} & * & \text{true}
\end{pmatrix}.
\]

Running HFL, \([W\text{INDY} \neq \text{true}]\) and \([O\text{UTLOOK} \neq \text{sunny}]\) which are equivalent to \([W\text{INDY} = \text{false}]\) and \([O\text{UTLOOK} = \text{rain}]\) respectively are both chosen as inevitable selectors and they cover all of the five rows in \(EMD_2\). Therefore, the second \(Hfl\) is

\([W\text{INDY} = \text{false}] \land [O\text{UTLOOK} = \text{rain}]\).

The third intersecting group is \(\{e_5^+\}\) and the disjunction matrix \(EMD_3\) is

\[
EMD_3 = \begin{pmatrix}
  \text{rain} & \text{hot} & \text{high} & \text{true} \\
  \text{rain} & \text{cool} & * & \text{true} \\
  * & \text{hot} & * & \text{true} \\
  * & * & \text{high} & * \\
  * & \text{hot} & \text{high} & * \\
  * & \text{cool} & * & * \\
  \text{rain} & * & * & \text{true}
\end{pmatrix}.
\]

When running HFL, \([T\text{EMPERATURE} \neq \{\text{hot, cool}\}]\) is first chosen as an inevitable selector and it covers rows 1, 2, 3, 5 and 6. Attributes \(O\text{UTLOOK}\) and \(H\text{UMIDITY}\) are then excluded by attribute \(W\text{INDY}\) and \([W\text{INDY} \neq \text{true}]\) is finally chosen as an inevitable selector on the fourth row after \(O\text{UTLOOK}\) and \(H\text{UMIDITY}\) have been crossed out. The equivalent \(Hfl\) for this intersecting group is

\([T\text{EMPERATURE} = \text{mild}] \land [W\text{INDY} = \text{false}]\).

Therefore,

\(^5\)This is because \([O\text{UTLOOK} = \text{overcast}]\) is shorter than \([O\text{UTLOOK} \neq \{\text{rain, sunny}\}]\) (see Section 4.1.2). Please also note that the grouping option in C4.5 and C4.5rules generates similar expressions (see Example 1 in Section 2).
\[ H_{cv} = [\text{OUTLOOK} = \text{overcast}] \land [\text{WINDY} = \text{false}] \land [\text{OUTLOOK} = \text{rain}] \land [\text{TEMPERATURE} = \text{mild}] \land [\text{WINDY} = \text{false}] \]

whose equivalent rule in variable-valued logic is:

\[
\begin{align*}
[ & \text{OUTLOOK}=\text{overcast} ] \\
\lor \\
[ & \text{WINDY}=\text{false} ] \\
[ & \text{OUTLOOK}=\text{rain} ] \\
\lor \\
[ & \text{TEMPERATURE}=\text{mild} ] \\
[ & \text{WINDY}=\text{false} ] \\
\rightarrow \\
[ & \text{DECISION}=\text{Play} ].
\end{align*}
\]

Meanwhile, the decision trees generated by C4.5 and ID3 on the example set in Table 3 are the same as given in Figure 4 which is converted by C4.5 rules into the rules below:

if \( \text{OUTLOOK} = \text{overcast} \) then \( \text{Play} \);
if \( \text{OUTLOOK} = \text{sunny} \) and \( \text{TEMPERATURE} = \text{hot} \) then \( \text{Don't Play} \);
if \( \text{OUTLOOK} = \text{sunny} \) and \( \text{TEMPERATURE} = \text{cool} \) then \( \text{Don't Play} \);
if \( \text{OUTLOOK} = \text{sunny} \) and \( \text{HUMIDITY} = \text{high} \) then \( \text{Don't Play} \);
if \( \text{OUTLOOK} = \text{sunny} \) and \( \text{TEMPERATURE} = \text{mild} \) and \( \text{HUMIDITY} = \text{normal} \) then \( \text{Play} \);
if \( \text{OUTLOOK} = \text{rain} \) and \( \text{WINDY} = \text{true} \) then \( \text{Don't Play} \); and
if \( \text{OUTLOOK} = \text{rain} \) and \( \text{WINDY} = \text{false} \) then \( \text{Play} \).

**Example 4.** Table 4 shows a set of training examples of more than two classes. Running HCV, we get the following rules for each of the three classes.

\[
\begin{align*}
[ & \text{X1} = [0] ] \\
[ & \text{X3} = [a] ] \\
\lor \\
[ & \text{X2} = [b] ] \\
[ & \text{X3} \leftrightarrow [b] ] \\
\rightarrow \\
\text{The C1 class}
\end{align*}
\]

\[
\begin{align*}
[ & \text{X2} = [c] ] \\
[ & \text{X4} = [1] ] \\
\lor \\
[ & \text{X1} = [1] ] \\
[ & \text{X2} \leftrightarrow [b] ] \\
\rightarrow \\
\text{The C3 class}
\end{align*}
\]

\[
\begin{align*}
[ & \text{X1} = [0] ] \\
[ & \text{X3} \leftrightarrow [a] ] \\
\rightarrow \\
\text{The C2 class}
\end{align*}
\]
Figure 4: A decision tree (by ID3 and C4.5) for Table 3

Figure 5: A decision tree (by ID3 and C4.5) for Table 4
Table 4: Cases of $C_1$, $C_2$ and $C_3$

<table>
<thead>
<tr>
<th>ORDER</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>C3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>1</td>
<td>C3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>C3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>C3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>b</td>
<td>c</td>
<td>1</td>
<td>C1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>b</td>
<td>b</td>
<td>0</td>
<td>C2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>C2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>0</td>
<td>C1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>C1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>c</td>
<td>c</td>
<td>0</td>
<td>C3</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>c</td>
<td>b</td>
<td>1</td>
<td>C3</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>c</td>
<td>b</td>
<td>0</td>
<td>C2</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>C1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>c</td>
<td>c</td>
<td>1</td>
<td>C3</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>c</td>
<td>a</td>
<td>0</td>
<td>C1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>0</td>
<td>C3</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>C1</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>C1</td>
</tr>
</tbody>
</table>

Meanwhile, the decision trees generated by C4.5 and ID3 on the example set in Table 4 are the same as given in Figure 4 which is converted by C4.5rules into the rules below:

If $X_1 = 0$ and $X_3 = a$ then class = $C_1$
If $X_1 = 1$ and $X_2 = b$ then class = $C_1$
If $X_1 = 0$ and $X_3 = b$ then class = $C_2$
If $X_1 = 0$ and $X_2 = a$ and $X_3 = c$ then class = $C_2$
If $X_2 = c$ and $X_3 = c$ then class = $C_3$
If $X_1 = 1$ and $X_2 = a$ then class = $C_3$
If $X_1 = 1$ and $X_2 = c$ then class = $C_3$

4.3 A Comparison Between HCV and AE1

As we mentioned in Section 3, the extension matrix approach was first introduced in AE1 in 1985 (Hong, 1985). To the best of the author’s knowledge, the approach itself has not been improved at all except in the author’s HCV algorithm. The developer of AE1 has developed an AE5 system (Hong, 1989a) based on AE1 but the basic algorithm remains the same. The only difference between AE1 and AE5 is that some facilities such as constructive and incremental induction have been added to the latter. The two major problems of AE1 described in Section 3.3 also apply to AE5. HCV is based on the extension matrix approach developed in AE1, but there are two radical improvements on the approach itself in HCV as well as those implementation aspects described in Section 4.4.

1. The use of disjunction matrices.

Although disjunction matrices are also defined in (Hong, 1985), AE1 does not produce and use them in the partitioning of positive examples. AE1 still needs to produce all extension matrices of positive examples against negative examples (by remembering only dead elements in each extension matrix (Hong, 1989a)), whereas in HCV, as seen in Section 4.4, we need at most two extension matrices at each time stage. By using disjunction matrices, HCV can save both a lot of data space and provide a natural way to reduce its rules’ complexity. As we have seen from Theorem 7, if there exists at least one conjunctive cover in a given training example
set, the formula produced by HCV must be a conjunctive one. This is by no means guaranteed in AE1.

2. Three complete strategies in HFL.

HCV has provided a reasonable solution to both the MFL and the MCV problems, described in Section 3.2, which are NP-hard in nature. So the second disadvantage of AE1 has been removed in HCV. Although the first disadvantage of AE1 still exists in HCV’s least-frequency strategy, we have provided three complete strategies at the same time. For those example sets where the three strategies are enough to produce the final results, we can guarantee that the results are optimal. According to all the experiments carried out thus far including those mentioned in this paper, the three strategies are always very useful even when they are not enough to produce an optimal result.

Neither AE1 nor AE5 is available for public experiments. They have provided no facilities for noise handling and discretization of real-valued attributes (see Sections 5 and 6). We were very worried about the second disadvantage mentioned in Section 3.3, when trying to implement AE1 in the HCV (Version 2.0) program. Therefore, AE1 and AE5 are not included in the experiments of Section 7, although one may think that making a comparison between AE1 and HCV in terms of run time and accuracy is sufficient to justify the benefit of the above improvements in HCV. Such a comparison is less significant than the comparison we are making in Section 7, because we have said in the introduction that the motivation for this paper is to take the newly-developed extension matrix approach and improve it to be competitive with ID3-like algorithms, the best-known family of induction algorithms.

4.4 Some Related Problems in Implementation

The HCV algorithm has been implemented on both PC machines (Wu, 1992c) and workstations (Wu, 1992b; Wu, 1995). Both implementations can classify more than two classes of examples and produce rules for each class of preclassified examples by assuming that examples not classified as positive are negative. For the example set in Table 5, which is equivalent to Table 1, the rules produced by HCV are shown below:

<table>
<thead>
<tr>
<th>ORDER</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>a</td>
<td>#</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>b</td>
<td>c</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>b</td>
<td>b</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>a</td>
<td>c</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>#</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>c</td>
<td>c</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>c</td>
<td>b</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>c</td>
<td>b</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>c</td>
<td>c</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>c</td>
<td>a</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>T</td>
</tr>
</tbody>
</table>

---

6In noisy domains we cannot expect the first three strategies to work without using the least-frequency strategy.

21
\[
\begin{align*}
\lor & \quad [X2 = [b]] \\
\lor & \quad [X1 = [0]] \\
\lor & \quad [X2 = [a]] \\
\rightarrow & \quad [X2<>[b]] \\
\lor & \quad [X1 = [1]]
\end{align*}
\]

The $T$ class.

Selectors or conjunctions in both implementations take the form of both $[X = S1]$ and $[X <> S2]$ (see Section 4.1.2), and choose whichever is the shorter. They also allow the user to evaluate the rules' accuracy in terms of a set of preclassified test examples (see Section 7).

### 4.4.1 Don't Cares in HCV

The \# symbol in HCV, like in many other induction programs, has a specific meaning when representing attribute values: Don't Cares.

A Don't Care value of an attribute in an example is used to indicate that the value of the attribute is irrelevant to the classification of the example. An example with Don't Care values can always be converted into a number of equivalent examples that have no Don't Cares. Therefore, a Don't Care attribute is universally quantified and Don't Care values can in practice be used to compress data spaces.

In the extension matrix approach, a dead element (*) in an extension matrix indicates that a positive example and a negative example have the same attribute value and therefore the attribute value cannot be used to distinguish the positive example from the negative example. When a negative example (say the $i$-th) has a Don't Care value on an attribute (say the $j$-th), the $r_{ij}$ in every positive example's extension matrix must be * because the negative example can take every positive example's $j$-th attribute value. When a positive example has a Don't Care value on its $j$-th attribute, all the values on the $j$-th column of the positive example's extension matrix must be * according to the meanings of Don't Care values and *_. Therefore, the \# values are processed at induction time in HCV in a very straightforward way.

### 4.4.2 Size of Extension Matrices

In each extension matrix or disjunction matrix, all nondead elements are the same as those in the negative example matrix $NEM$ according to Definitions 1 and 3. We only need to remember the dead elements in execution. The number of dead elements in the extension matrix or disjunction matrix must be less than $na$ where $n$ and $a$ are the numbers of negative examples and attributes in a given example set.

All the positive examples are processed one by one in the HCV algorithm. At each stage, we need at most one extension matrix and one disjunction matrix. The space needed for an implementation of the HCV algorithm is less than $2na$. Therefore, the HCV algorithm is low-order polynomial learnable in both time and space. This is an important feature for good induction algorithms (Valiant, 1984).

### 5 Noise Handling with HCV

As argued in the introduction, noise handling is as important as induction itself in data mining. This section presents some of the noise handling techniques developed and implemented in the HCV (Version 2.0) software that are unique with the extension matrix approach. (Wu, Krisir, & Måhlén, 1995) provides more details of all noise handling methods implemented in the HCV.
5.1 Approximate partitioning

HCV first partitions the positive examples of a concept into a number of groups, and then forms a description for each group. Partitioning in HCV is one of the most fundamental steps. In noisy environments, we can allow all the examples in each group to approximately (rather than strictly) meet some criteria. In HCV (Version 2.0), once there is a path which can pass through most rows of the disjunction matrix, we can put all the concerned positive examples into an intersecting group. There is a specific strategy which puts some positive examples into a partition even knowing that the partition is not going to be an exact intersecting group. Two parameters are attached to this strategy, to tell the maximum acceptable number of dead rows (rows with dead elements only) in a disjunction matrix when a positive example is joining a partition, and the maximum number of negative examples which the rules of positive examples are allowed to cover.

5.2 Stopping criteria

To avoid overfitting, we can use certain stopping criteria to determine when to cease forming more details of a concept description. When constructing a description for a partition, we do not need to make the description 100% consistent with the positive examples, but instead, cover most of them.

In the case of decision tree construction, if most of the examples at one node belong to a specific class, we can stop exploring the node any further by simply assigning it to the specific class and treat it as a leaf. This is what we call pre-pruning of decision trees. Pre-pruning has been used extensively with ID3-like algorithms (Cestnik, Kononenko, & Bratko, 1987). A number of criteria including the $\chi^2$-test pre-pruning (Quinlan, 1993) and the information-based stopping criterion (Quinlan, 1990) have been designed. Concept descriptions produced this way do not fit all the examples in the training set. This is what we call underfitting. Overfitting and underfitting are two opposite operations in induction. Normally, the user conducting the experiments or running the induction system on their specific data sets is expected to specify a threshold value $\alpha$ to indicate that they do not require the concept descriptions to be 100% but $(100 - \alpha)\%$ consistent with the training examples.

In HCV (Version 2.0), a switch is provided for the user to specify a $\alpha$ value for the HFL algorithm (see Section 4.1) to avoid overfitting. The $\alpha$ switch looks similar to the $-c$ option in C4.5 and C4.5rules, but there are two differences. First, the $-c$ option in C4.5 and C4.5rules deals with some positive examples and some negative examples at a decision node, whereby the $\alpha$ switch in HFL deals with a partition of positive examples against all negative examples. Second, the $\alpha$ value in HCV (Version 2.0) affects the length of the conjunctive rules produced by HFL, but the $-c$ option changes the number of leaf nodes of a decision tree in C4.5 and subsequently the number of rules in C4.5rules.

6 Dealing with Real-Valued Attributes

In the context of rule induction and decision tree construction, dealing with a continuous domain means discretization of the numerical domain into a certain number of intervals (Dougherty, Kohavi, & Sahami, 1995; Pfahringer, 1995). The discretized intervals can be treated in a similar way to nominal values during induction and deduction. The most difficult aspect of discretization is to find the right places to set up interval borders. There are a number of different methods available in HCV (Version 2.0). The user can choose between them by using the options provided.

---

7Email xindong@insect.sd.monash.edu.au to obtain a copy of the software and its manual.
by the software. Below we detail the default method in HCV (Version 2.0) for the experiments in Section 7.

6.1 The information gain heuristic

When the examples in the training set have taken values of \(x_1, \ldots, x_n\) in ascending order on a continuous attribute, we can use the information gain (or entropy) heuristic adopted in ID3 (Quinlan, 1986) to find the most informative border to split the value domain of the continuous attribute. (Fayyad & Irani, 1992) have shown that the maximum information gain by the heuristic is always achieved at a cut point (say, the mid-point) between the values taken by two examples of different classes.

The information gain based discretization has been reported in a number of references (e.g., (Dougherty, Kohavi, & Sahami, 1995)) to perform better than other methods, and therefore it is implemented as the default discretization method in HCV (Version 2.0). We adopt the information gain heuristic in the following way: Each \(x = (x_i + x_{i+1})/2 (i = 1, \ldots, n - 1)\) is a possible cut point if \(x_i\) and \(x_{i+1}\) have taken by examples of different classes in the training set; use the information gain heuristic to check each of the possible cut points and find the best split point; then run the same process on the left and right halves of the splitting to split them further. The number of intervals produced this way may be very large if the attribute is not very informative. The following criteria are adopted from (Catlett, 1991) to stop the recursive splitting:

- Stop if the information gain on all cut points is the same,
- Stop if the number of examples to split is less than a certain number (e.g. fourteen), and
- Limit the number of intervals to be produced to a certain number (e.g. eight).

7 A Performance Comparison of HCV with Other Induction Algorithms

HCV has been tested on tens of example sets including those outlined below. All the experiments the author has conducted have shown empirically that the rules produced by HCV in variable-valued logic are significantly more compact than the decision trees produced by ID3-like algorithms in terms of the numbers of conjunctive rules and conjunctions when the decision trees are transformed into decision rules. However, since HCV rules permit membership in their conjunctions whereby ID3 (Quinlan, 1986) and C4.5 (without value grouping) (Quinlan, 1993) do not, one may ask whether HCV rules are smaller simply because of the representational difference. The answer is that permitting complex conjunctions normally helps, but this is not the whole problem. Some ID3-like algorithms like NewID (Boswell, 1990) always perform value grouping, and others like C4.5 have implemented value grouping as an option (see Figure 2 in Section 2), but their complexity is still high. To demonstrate this, ID3, NewID, C4.5 and C4.5 rules are selected from the ID3-like algorithms in this section to compete with HCV. Value grouping is carried out in NewID, C4.5 with grouping and C4.5 rules with grouping (see Tables 5 and 6).

Other algorithms, such as GREEDY3 (Pagallo & Haussler, 1990), CN2 (Clark, & Niblett, 1989), AQ15 (Michalski et al., 1986) and CART (Breiman et al., 1984), are good at processing noisy data, but have not provided efficient mechanisms to carry out both discretization of continuous domains and process nominal values. This is not say that a comparison between them with HCV is impossible; the author is planning to add these facilities to these algorithms for further experimentation in the near future. However, the aim of this paper is not to demonstrate that HCV performs better than all existing algorithms in any sense, but just to demonstrate
that the extension matrix approach, after improvements in HCV, is competitive with ID3-like algorithms, the best-known family of induction algorithms. When comparing different algorithms of different families, such as C4.5 from the ID3 family and AQ15 from the AQ family, we need to develop a set of meaningful metrics in terms of induction time, size of rules, the time to execute these rules and the rules’ predictive accuracy, and we can always expect that different algorithms will perform better on different metrics.

Sections 7.1 and 7.2 provide a performance comparison of HCV with ID3, C4.5, C4.5rules and NewID in terms of rule compactness and accuracy on the three MONK’s problems (Thrun et al., 1991). Section 7.3 presents summaries of experiments with HCV, C4.5, C4.5rules and NewID on more data sets.

The HCV rules reported in Section 7.2 were produced by the HCV (Version 1.0) software (Wu, 1992b) which always tries to produce consistent rules with the training sets. ID3, C4.5, C4.5rules and NewID were also conducted to produce as consistent induction results as possible with the training sets in Section 7.2. These results can be improved by using noise handling facilities, and are outlined in Section 7.3 in terms of accuracy.

7.1 Example 5: The MONK’s Problems

The MONK’s problems are derived from an artificial robot domain, in which robots (examples) are described by six multiple-valued attributes, i.e., head_shape, body_shape, is_smiling, holding, jacket_color, and has_tie. The size of the value sets of the six attributes are 3, 3, 2, 3, 4 and 2, respectively as below.

\[
\begin{align*}
\text{head\_shape} &\in \{\text{round, square, octagon}\} \\
\text{body\_shape} &\in \{\text{round, square, octagon}\} \\
\text{is\_smiling} &\in \{\text{yes, no}\} \\
\text{holding} &\in \{\text{sword, balloon, flag}\} \\
\text{jacket\_color} &\in \{\text{red, yellow, green, blue}\} \\
\text{has\_tie} &\in \{\text{yes, no}\}
\end{align*}
\]

Consequently, the example space consists of the total of $3\times3\times2\times3\times4\times2=432$ possible examples. The three MONK’s problems, called $M1$, $M2$, and $M3$, are all binary classifications defined over the same space. They differ in the type of the concept to be learned, and in the amount of noise in the training examples. Each problem is given by a logical description of a concept. Robots belong to either this concept or not. Instead of providing a complete concept description to the induction problem, only a subset of all 432 possible robots with its classification is given. The induction task is then to generalize over these examples and, if the particular induction technique at hand allows this, to derive a simple concept description. After a concept description has been produced by a induction algorithm from the training examples of each of the three problems, the whole 432 examples are used to test the accuracy of the concept description.

The three MONK’s problems are specifically designed as below:

- Problem M1: \((\text{head\_shape} = \text{body\_shape}) \ OR (\text{jacket\_color} = \text{red})\)

From 432 examples (referred to as Test Set 1 hereafter, 216 positive and 216 negative), 124 (62 positive and 62 negative) were randomly selected for the training set (referred to as Training Set 1 hereafter). There were no misclassifications in Training Set 1.

This problem is in standard disjunctive normal form (DNF) and is supposed to be easily learnable by all symbolic induction algorithms.
• Problem M2: exactly two of the six attributes have their first value

From 432 examples (referred to as Test Set 2 hereafter, 142 positive and 190 negative), 169 (64 positive and 105 negative) were randomly selected as training examples (referred to as Training Set 2 hereafter). Again, there was no noise in Training Set 2.

This problem is said to be among the most difficult to learn using sole logic-based inductive learners (such as AQ11-like algorithms and ID3-like algorithms). It combines different attributes in a way which makes it complicated to describe in DNF or CNF (conjunctive normal form) using the given attributes only.

• Problem M3: (jacket_color = green) AND (holding = sword) OR (jacket_color is NOT blue) AND (body_shape is NOT octagon)

From 432 examples (referred to as Test Set 3 hereafter, 228 positive and 204 negative), 122 (60 positive and 62 negative) were selected randomly, and among them there were 5% misclassifications, i.e., noise in the training set (referred to as Training Set 3 hereafter).

This problem is again in DNF but serves to evaluate induction algorithms in the presence of noise.

7.2 Performance comparison on the MONK’s problems

As each leaf in a decision tree corresponds to a conjunctive decision-tree-traversal rule, the number of leaves in a decision tree is equivalent to the number of conjunctive decision-tree-traversal rules. Table 6 lists the numbers of conjunctive rules and the numbers of conditions within these rules produced by different induction algorithms including HCV\(^8\). As mentioned at the beginning of Section 7, the decision trees and rules produced by C4.5 with grouping, C4.5rules with grouping, NewID and HCV all contain grouped values in their conjunctions.

From Table 6, we can see that the rules produced by HCV are the most compact in terms of the numbers of conjunctive rules and conjunctions. The only exception is that the rules produced by C4.5rules with grouping are smaller than HCV.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Training Set 1</th>
<th>Training Set 2</th>
<th>Training Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rules</td>
<td>conditions</td>
<td>rules</td>
</tr>
<tr>
<td>ID3</td>
<td>53</td>
<td>216</td>
<td>105</td>
</tr>
<tr>
<td>C4.5</td>
<td>60</td>
<td>262</td>
<td>113</td>
</tr>
<tr>
<td>C4.5 with grouping</td>
<td>9</td>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>C4.5rules</td>
<td>31</td>
<td>101</td>
<td>97</td>
</tr>
<tr>
<td>C4.5rules with grouping</td>
<td>8</td>
<td>19</td>
<td>46</td>
</tr>
<tr>
<td>NewID</td>
<td>21</td>
<td>143</td>
<td>59</td>
</tr>
<tr>
<td>HCV</td>
<td>7</td>
<td>16</td>
<td>39</td>
</tr>
</tbody>
</table>

Table 7 provides numerical evaluation of the accuracy of the rules or decision trees produced by ID3, C4.5, C4.5rules, NewID and HCV based on the percentage of test examples correctly classified. As we can see, HCV works perfectly well on the M1 problem. Its accuracy is the second best on the M2 problem. In noisy environments like the M3 problem, the HCV (Version 1.0) software does as well as C4.5 although HCV (Version 1.0) has no specific noise handling facilities.

\(^8\)To be fair, we have counted HCV rules for both positive and negative examples. This is also the case with the experiment summaries in Section 7.3.
Table 7: Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Test Set 1</th>
<th>Test Set 2</th>
<th>Test Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3</td>
<td>83.3%</td>
<td>68.3%</td>
<td>94.4%</td>
</tr>
<tr>
<td>C4.5</td>
<td>82.4%</td>
<td>69.7%</td>
<td>90.3%</td>
</tr>
<tr>
<td>C4.5 with grouping</td>
<td>100%</td>
<td>82.4%</td>
<td>93.1%</td>
</tr>
<tr>
<td>C4.5rules</td>
<td>92.4%</td>
<td>75.7%</td>
<td>85.4%</td>
</tr>
<tr>
<td>C4.5rules with grouping</td>
<td>100%</td>
<td>81.0%</td>
<td>91.4%</td>
</tr>
<tr>
<td>NewID</td>
<td>93%</td>
<td>78%</td>
<td>89%</td>
</tr>
<tr>
<td>HCV</td>
<td><strong>100.00%</strong></td>
<td><strong>81.7%</strong></td>
<td><strong>90.3%</strong></td>
</tr>
</tbody>
</table>

In the above experiments with the MONK's problems, C4.5 with grouping and C4.5rules with grouping have both performed pruning while HCV has not. For example, the accuracy results for C4.5 with grouping before pruning were 96.8%, 75.5% (which is worse than HCV's accuracy) and 93.1% respectively on the test sets of the three MONK's problems. C4.5rules with grouping takes the pruned results of C4.5 with grouping as the input and performs its own pruning when decomposing a decision tree into a set of rules. HCV's results are improved significantly with HCV (Version 2.0) (see Table 9), which provides pruning and noise handling facilities as outlined in Section 5.

7.3 Summaries of more experiments

This section contains summaries of experiments with HCV (Version 2.0), C4.5, C4.5rules and NewID on more databases.

7.3.1 The Data

The databases used in our experiments (see Table 8) can be divided into four groups. The first group is made up of data with 100% nominal attributes. The second group contains data of mixed nominal and continuous attributes. In turn, the third group is made up of data with 100% continuous attributes. These three groups of databases were all copied from the University of California at Irvine machine learning database repository (Murphy & Aha, 1995). The majority of these databases were obtained from real world domains and are noisy.

The fourth group of databases originates from an aluminium smelter and was obtained by process sensors monitoring the aluminium production process (Wu, Urpani, & Sykes, 1996). The first database in this group, UFT, predicts different abnormal process conditions based on sensor data. The last three databases Prediction 1, 2 and 3 augment the original sensor data through the use of derived variables (such as rate of change and differences) in an attempt to predict one, two and three days ahead whether a production cell will exhibit abnormal operating conditions. All these databases are very noisy and not guaranteed to be complete in terms of the given attributes.

These databases have been selected because each of them consists of two standard components when created or collected by the original providers: a training set and a test set. The standard deviation has been given in some by the original database providers. The databases have been used "as is". Example ordering has not been changed, neither have examples been moved between the sets. For each database, we ran each of HCV (Version 2.0), C4.5, C4.5rules and NewID on the training set, and the accuracies listed in Tables 8 are from the test set.

7.3.2 Test Conditions

Throughout the experiments, default parameters were adopted for all the four programs, C4.5, C4.5rules, NewID and HCV (Version 2.0) as recommended by the respective authors, for all databases. The default parameters for each algorithm are very complicated, as they include parameters for data preprocessing, induction, postprocessing (or postpruning) of induction, and
Table 8: Databases Characteristics

<table>
<thead>
<tr>
<th>Database</th>
<th># of Instances</th>
<th>Attributes</th>
<th>Classes</th>
<th>Majority Class (%)</th>
<th>Continuous Attributes (%)</th>
<th>Avg # of Values per Attributes</th>
<th>Unknown Values (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auditology</td>
<td>226</td>
<td>69</td>
<td>24</td>
<td>21.00</td>
<td>0.00</td>
<td>2.23</td>
<td>200</td>
</tr>
<tr>
<td>Hayes-Roth</td>
<td>160</td>
<td>4</td>
<td>3</td>
<td>40.60</td>
<td>0.00</td>
<td>4.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Monk1</td>
<td>556</td>
<td>6</td>
<td>2</td>
<td>50.00</td>
<td>0.00</td>
<td>2.80</td>
<td>0.00</td>
</tr>
<tr>
<td>Monk2</td>
<td>601</td>
<td>6</td>
<td>2</td>
<td>65.70</td>
<td>0.00</td>
<td>2.80</td>
<td>0.00</td>
</tr>
<tr>
<td>Monk3</td>
<td>554</td>
<td>6</td>
<td>2</td>
<td>52.00</td>
<td>0.00</td>
<td>2.80</td>
<td>0.00</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>958</td>
<td>9</td>
<td>2</td>
<td>65.30</td>
<td>0.00</td>
<td>3.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Soybean</td>
<td>683</td>
<td>35</td>
<td>19</td>
<td>13.50</td>
<td>0.00</td>
<td>2.80</td>
<td>12.00</td>
</tr>
<tr>
<td>Vote</td>
<td>435</td>
<td>16</td>
<td>2</td>
<td>61.40</td>
<td>0.00</td>
<td>3.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Anneal</td>
<td>798</td>
<td>38</td>
<td>6</td>
<td>76.00</td>
<td>16.00</td>
<td>3.32</td>
<td>64</td>
</tr>
<tr>
<td>Aus-Credit</td>
<td>690</td>
<td>15</td>
<td>2</td>
<td>56.00</td>
<td>40.00</td>
<td>4.56</td>
<td>0.65</td>
</tr>
<tr>
<td>Cleveland 2</td>
<td>303</td>
<td>13</td>
<td>5</td>
<td>54.00</td>
<td>38.00</td>
<td>3.00</td>
<td>2.00</td>
</tr>
<tr>
<td>Cleveland 5</td>
<td>303</td>
<td>13</td>
<td>5</td>
<td>54.00</td>
<td>38.00</td>
<td>3.00</td>
<td>2.00</td>
</tr>
<tr>
<td>Hungarian 2</td>
<td>294</td>
<td>13</td>
<td>5</td>
<td>64.00</td>
<td>38.00</td>
<td>3.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>3572</td>
<td>29</td>
<td>5</td>
<td>92.00</td>
<td>24.10</td>
<td>2.18</td>
<td>5.52</td>
</tr>
<tr>
<td>Imports 85</td>
<td>205</td>
<td>25</td>
<td>7</td>
<td>32.70</td>
<td>60.00</td>
<td>6.00</td>
<td>1.15</td>
</tr>
<tr>
<td>Lab Neg</td>
<td>56</td>
<td>16</td>
<td>2</td>
<td>65.00</td>
<td>50.00</td>
<td>2.62</td>
<td>35.75</td>
</tr>
<tr>
<td>Swiss 2</td>
<td>123</td>
<td>13</td>
<td>5</td>
<td>39.00</td>
<td>38.00</td>
<td>3.00</td>
<td>17.00</td>
</tr>
<tr>
<td>Swiss 5</td>
<td>123</td>
<td>13</td>
<td>5</td>
<td>39.00</td>
<td>38.00</td>
<td>3.00</td>
<td>17.00</td>
</tr>
<tr>
<td>Va 2</td>
<td>200</td>
<td>13</td>
<td>5</td>
<td>28.00</td>
<td>38.00</td>
<td>3.00</td>
<td>26.40</td>
</tr>
<tr>
<td>Va 5</td>
<td>200</td>
<td>13</td>
<td>5</td>
<td>28.00</td>
<td>38.00</td>
<td>3.00</td>
<td>26.40</td>
</tr>
<tr>
<td>Bupa</td>
<td>345</td>
<td>7</td>
<td>2</td>
<td>55.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>9</td>
<td>7</td>
<td>36.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>(without ID number)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ionosphere</td>
<td>351</td>
<td>34</td>
<td>2</td>
<td>64.10</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Pima</td>
<td>768</td>
<td>8</td>
<td>2</td>
<td>65.10</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Wine</td>
<td>178</td>
<td>13</td>
<td>3</td>
<td>40.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>UPT</td>
<td>407</td>
<td>20</td>
<td>2</td>
<td>50.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Prediction 1</td>
<td>721</td>
<td>56</td>
<td>2</td>
<td>50.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Prediction 2</td>
<td>721</td>
<td>68</td>
<td>2</td>
<td>50.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
<tr>
<td>Prediction 3</td>
<td>616</td>
<td>68</td>
<td>2</td>
<td>50.00</td>
<td>100.00</td>
<td>n/a</td>
<td>0.00</td>
</tr>
</tbody>
</table>

deduction of induction results. In HCV (Version 2.0), for example, there are tens of parameters. The reader is referred to the corresponding references for details of these default parameters. Pruning is used in all of these programs by default. The results shown for C4.5 and NewID are the pruned ones which are generally better than the unpruned ones in our experiments.

Obviously fine tuning different parameters in HCV (Version 2.0) would have achieved higher accuracy rates. This however would have been at the expense of a loss in generality and applicability of the conclusions.

For databases with continuous attributes, the discretization method adopted in C4.5, C4.5rules and NewID is the information gain heuristic outlined in Section 6.1, which is also the default method for HCV (Version 2.0).

Although all these algorithms/programs used in the experiments are heuristic, (e.g., when the order of attributes changes the induction results could be different), each of them has its own deterministic strategies at non-deterministic choice points, so multiple runs of the same program on the same database always produce the same results. Therefore, cross-validation is not needed in our experiments.

7.3.3 Accuracy Results

Table 9 shows results produced by HCV, C4.5, C4.5rules and NewID by using their default parameters. The best result for each problem is highlighted with **boldface** font in the table.

Among the 8 databases in the first group, which contain no real-valued attributes, HCV gives the best results for 5, C4.5 and C4.5rules each for 2, and NewID for only one. The sum is greater than 8 because different programs get the same results for some of the databases.

On the 12 databases in the second group, each of which contains both nominal and continuous attributes, C4.5 has the best results for 6, C4.5rules for 3, NewID for only one, and HCV gets
Table 9: Experiments with Default Parameters

<table>
<thead>
<tr>
<th>Database</th>
<th>C4.5</th>
<th>C4.5rules</th>
<th>NewID</th>
<th>HCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audiology</td>
<td>84.6%</td>
<td>81.8%</td>
<td>85.0%</td>
<td>80.8%</td>
</tr>
<tr>
<td>Hayes-Roth</td>
<td>75.7%</td>
<td>71.4%</td>
<td>79.0%</td>
<td>85.7%</td>
</tr>
<tr>
<td>Monk1</td>
<td>75.7%</td>
<td>100%</td>
<td>93.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Monk2</td>
<td>65.0%</td>
<td>65.3%</td>
<td>78.0%</td>
<td>85.2%</td>
</tr>
<tr>
<td>Monk3</td>
<td>97.2%</td>
<td>96.3%</td>
<td>97.0%</td>
<td>98.1%</td>
</tr>
<tr>
<td>Soybean</td>
<td>82.4%</td>
<td>81.6%</td>
<td>66.0%</td>
<td>80.2%</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>84.7%</td>
<td>100%</td>
<td>84.0%</td>
<td>88.0%</td>
</tr>
<tr>
<td>Vote</td>
<td>97.0%</td>
<td>95.6%</td>
<td>96.0%</td>
<td>97.8%</td>
</tr>
<tr>
<td>Anneal</td>
<td>93.0%</td>
<td>91.0%</td>
<td>76.0%</td>
<td>98.0%</td>
</tr>
<tr>
<td>Cleveland 2</td>
<td>76.9%</td>
<td>76.9%</td>
<td>67.0%</td>
<td>78.0%</td>
</tr>
<tr>
<td>Cleveland 5</td>
<td>56.0%</td>
<td>57.1%</td>
<td>43.0%</td>
<td>56.0%</td>
</tr>
<tr>
<td>Crx</td>
<td>80.0%</td>
<td>80.5%</td>
<td>79.0%</td>
<td>82.5%</td>
</tr>
<tr>
<td>Hungarian 2</td>
<td>80.0%</td>
<td>85.0%</td>
<td>78.0%</td>
<td>86.3%</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>99.4%</td>
<td>99.4%</td>
<td>92.0%</td>
<td>97.8%</td>
</tr>
<tr>
<td>Imports 85</td>
<td>67.8%</td>
<td>67.8%</td>
<td>61.0%</td>
<td>62.7%</td>
</tr>
<tr>
<td>Labor Neg</td>
<td>82.4%</td>
<td>88.2%</td>
<td>65.0%</td>
<td>76.5%</td>
</tr>
<tr>
<td>Swiss 2</td>
<td>96.9%</td>
<td>87.5%</td>
<td>97.0%</td>
<td>96.9%</td>
</tr>
<tr>
<td>Swiss 5</td>
<td>31.2%</td>
<td>28.1%</td>
<td>22.0%</td>
<td>28.1%</td>
</tr>
<tr>
<td>Va 2</td>
<td>70.4%</td>
<td>71.8%</td>
<td>77.0%</td>
<td>78.9%</td>
</tr>
<tr>
<td>Va 5</td>
<td>26.8%</td>
<td>21.2%</td>
<td>20.0%</td>
<td>26.8%</td>
</tr>
<tr>
<td>Bupa</td>
<td>61.0%</td>
<td>61.0%</td>
<td>73.0%</td>
<td>57.6%</td>
</tr>
<tr>
<td>Glass (without ID number)</td>
<td>64.6%</td>
<td>85.9%</td>
<td>66.0%</td>
<td>72.3%</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>85.5%</td>
<td>85.5%</td>
<td>82.0%</td>
<td>88.0%</td>
</tr>
<tr>
<td>Pima</td>
<td>75.5%</td>
<td>75.9%</td>
<td>73.0%</td>
<td>73.9%</td>
</tr>
<tr>
<td>Wine</td>
<td>82.7%</td>
<td>82.7%</td>
<td>90.0%</td>
<td>90.4%</td>
</tr>
<tr>
<td>UFT</td>
<td>69.6%</td>
<td>72.1%</td>
<td>69.0%</td>
<td>70.37%</td>
</tr>
<tr>
<td>Prediction 1</td>
<td>68.6%</td>
<td>75.3%</td>
<td>78.0%</td>
<td>74.90%</td>
</tr>
<tr>
<td>Prediction 2</td>
<td>57.9%</td>
<td>57.5%</td>
<td>62.0%</td>
<td>67.78%</td>
</tr>
<tr>
<td>Prediction 3</td>
<td>53.8%</td>
<td>54.6%</td>
<td>62.0%</td>
<td>60.50%</td>
</tr>
</tbody>
</table>

the best results for 7.

Out of the 9 databases of the last two groups, which contain continuous data only, C4.5rules and HCV each obtained the best results on 3 of them. C4.5 performed rather badly on these databases, while NewID produced particularly good results on the average: it has the best results on 4 of the databases.

From these results, it is hard to generalize which algorithm performs better than others. Although HCV (Version 2.0) obtained the best results on more databases than any other algorithm, we still cannot conclude that it will perform better on other databases, because as shown in Table 9, there are other algorithms in each of these groups of databases that performed better than HCV. However, these experiments have demonstrated that HCV is competitive with ID3-like algorithms on all types of databases.

8 Conclusions

The HCV algorithm described in Section 4 is the most recent successor of the extension matrix based family of inductive algorithms (Wu, 1993b) for data mining. As its time is low-order polynomial, it is can be seen as one of the fastest data mining algorithms to date\(^9\). The rules

\(^9\)As mentioned in Section 4.5, the class of low-order polynomial algorithms gives a wide range of CPU-time performance on practical tasks, and we have found that HCV is much faster than C4.5 on some example sets and
generated in HCV take the form of variable-valued logic rules rather than decision trees.

The rules produced by HCV for each concept from an input database can be thought of as a disjunctive set of conjunctive terms. The partitioning technique adopted in HCV is a kind of "greedy covering" (Carbonell, 1990). So, HCV has attacked the best-known problem (induction of disjunctive concepts) in induction by coupling one of the oldest and best-known techniques, greedy covering, with the HFL algorithm which chooses the candidate conjuncts at each point in the run. The combination can guarantee that the result of HCV is a conjunctive rule for a concept, if there exists such a conjunctive rule for the concept, in a given training example set. This is a significant theoretical achievement in mining disjunctive concepts (Wu, 1993a). None of the ID3-like algorithms have attained this property.

Noise handling and discretization of continuous domains are two very important areas of data mining. In Section 5, we have introduced some of the noise handling techniques developed in HCV (Version 2.0). In Section 6, we dealt with discretization in particular.

In addition to the theoretical analysis, this paper has provided a battery of experimental results in Section 7 showing empirically that the rules generated by HCV are more compact than the decision trees or rules produced by ID3-like algorithms, and its predicative accuracy is competitive with ID3-like algorithms.

Acknowledgements

The author is indebted to Alex Gammermann (Royal Holloway, University of London), Johan Krisár (Royal Institute of Technology, Stockholm), Petter Måhlén (Royal Institute of Technology, Stockholm), Chris Mellish (Edinburgh University), Tim Niblett (Turing Institute), Robert Rae (Edinburgh University), Simon Raik (Monash University), Dave Robertson (Edinburgh University), Peter Ross (Edinburgh University), and Derek Sleeman (Aberdeen University), and the four anonymous reviewers for their extensive comments and advice on earlier versions of this paper.

References


the reverse is the case with other example sets. Therefore, we do not claim that HCV is faster than any other low-order polynomial algorithm in general.


