Machine Learning, 25, 23–50 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Using the Minimum Description Length Principle to Infer Reduced Ordered Decision Graphs

ARLINDO L. OLIVEIRA aml@inesc.pt
IST/INESC, R. Alves Redol 9, Lisboa, Portugal

ALBERTO SANGIOVANNI-VINCENTELLI alberto@eecs.berkeley.edu
Department of EECS, UC Berkeley, Berkeley, CA 94720

Editor: Jude Shavlik

Abstract. We propose an algorithm for the inference of decision graphs from a set of labeled instances. In particular, we propose to infer decision graphs where the variables can only be tested in accordance with a given order and no redundant nodes exist. This type of graphs, reduced ordered decision graphs, can be used as canonical representations of Boolean functions and can be manipulated using algorithms developed for that purpose. This work proposes a local optimization algorithm that generates compact decision graphs by performing local changes in an existing graph until a minimum is reached. The algorithm uses Rissanen's minimum description length principle to control the tradeoff between accuracy in the training set and complexity of the description. Techniques for the selection of the initial decision graph and for the selection of an appropriate ordering of the variables are also presented. Experimental results obtained using this algorithm in two sets of examples are presented and analyzed.

Keywords: Inductive learning, MDL principle, decision trees

1. Introduction

This paper describes heuristic algorithms for the induction of minimal complexity decision graphs from training set data. Decision graphs can be viewed as a generalization of decision trees, a very successful approach for the inference of classification rules (Breiman et al., 1984; Quinlan, 1986). The selection of decision graphs instead of decision trees, as the representation scheme, is important because very large decision trees are required to represent some concepts of interest. This makes it hard to learn these concepts using decision trees as the underlying representation. In particular, the quality of the generalization performed by a decision tree induced from data suffers because of two well known problems: the replication of subtrees required to represent some concepts and the rapid fragmentation of the training set data when attributes that can take a high number of values are tested at a node (Oliver, 1993). Reduced ordered decision graphs, the particular representation addressed in this work, are decision graphs that have no redundant nodes and where the tests performed on the variables follow some fixed order for all paths in the graph. These graphs exhibit some specific characteristics that makes them specially useful for the task at hand.

Decision graphs have been proposed as one way to alleviate these problems, but the algorithms proposed to date for the construction of these graphs suffer from serious limitations. Mahoney and Mooney (1991) proposed to identify related subtrees in a decision tree obtained using standard methods, but reported limited success since they observed no improvement in the quality of the generalization performed. Their lack of success may be partially explained by the fact that they used a non-canonical representation of Boolean functions (DNF expressions) to represent the functions implemented by these subtrees. The non-canonicity of this representation makes it a non-trivial process to identify identical subtrees and may explain, at least partially, the lack of effectiveness of the approach. Oliver (1993) proposed a greedy algorithm that performs either a join or a split operation, depending on which one reduces the description length to a larger extent. He reported improvements over the use of decision trees on relatively simple problems, but our experiments using a similar approach failed on more complex test cases because the algorithm tends to perform premature joins on complex problems.

Kohavi (1994) proposed an approach that also uses reduced ordered decision graphs. His approach builds an ordered decision graph in a bottom-up fashion, starting at the level closest to the terminal nodes. This choice is partially based on the fact that the widest level (i.e., the level with the larger number of nodes) of a reduced ordered decision graph is, in general, closer to the bottom of the graph. He uses this characteristic of reduced ordered decision graphs to argue that it is advantageous to select the variable ordering in a bottom-up fashion. This algorithm is, however, too inefficient to be used directly in large problems, although this limitation can be circumvented by the use of attribute-selection techniques. More recently, Kohavi (1995) proposed an alternative that is also based on the identification of common subtrees in a decision tree. However, unlike other approaches, this decision tree is constrained to exhibit the same ordering of tests for all possible paths in the tree, thereby suffering from potential data fragmentation. A combination of these approaches with some of the techniques introduced in this work may be worth studying.

The approach described in this paper is radically different. First, the problem of selecting an appropriate ordering for the variables is solved using one of the highly effective heuristic algorithms for variable reordering proposed in the logic synthesis literature (Rudell, 1993). Second, the algorithm that derives a compact decision graph uses many of the techniques developed in the logic synthesis and machine learning communities in its search for compact decision graphs. A compact decision graph is derived by performing incremental changes in an existing solution until a local optimum is obtained. The initial decision graph is obtained using the facilities provided by standard packages for the manipulation of decision graphs together with well-known machine learning techniques.

As with many other approaches to the induction of classification rules, the induction of a decision graph from a labeled training set has to deal with the problem of trading off accuracy in the training set with accuracy on future examples. In general, accuracy on unseen examples is closely related with the compactness of the description selected. Descriptions that are too complex may exhibit high accuracy in the training set data but will generalize poorly, a phenomenon commonly described as *over-fitting*. The general bias for simpler hypotheses is known as Occam's razor and has been addressed by a number of authors (Pearl, 1978; Blumer et al., 1986; Blumer et al., 1987). A particularly useful way of viewing this problem, the minimum description length (MDL) principle, has been proposed by Rissanen (1978,1986). Using this principle, the problem of selecting a decision graph that exhibits a good generalization accuracy can be formulated as the selection of a graph that minimizes the code length required to describe both the graph and the exceptions to that graph present in the training set data.

The selection of decision graphs as the representation scheme and the search for decision graphs of small description length can be viewed as the selection of a particular inductive bias. No bias is intrinsically superior to any other bias for all concepts, and an argument for the superiority of a particular bias can only be made in the context of a particular set of induction problems. A particularly clear study of the inherent equivalence of all biases in the absence of a context was presented by Schaffer (1994). He shows that any improvement produced by a particular algorithm in some set of problems has to be compensated by reduced performance in another set. We argue, however, that the bias for small decision graphs is appropriate for many interesting concepts and present empirical results that show that this is indeed the case for some types of practical problems.

As mentioned, this work draws heavily on techniques developed by other authors in the machine learning and logic synthesis fields. From machine learning, we use many of the techniques developed for the induction of decision trees (Quinlan, 1986) as well as the constructive induction algorithms first studied by Pagallo and Haussler (1990). From the logic synthesis field, we use the vast array of techniques developed for the manipulation of reduced ordered decision graphs as canonical representations for Boolean functions (Bryant, 1986; Brace et al., 1989) and the variable reordering algorithms studied by a number of different authors (Friedman and Supowit, 1990; Rudell, 1993). For the benefit of readers not familiar with the use of reduced ordered decision graphs as a tool for the manipulation of Boolean functions, Appendix A gives an overview of the techniques available and their relation to this work.

The remainder of this article is organized as follows: Section 2 introduces the basic concepts and definitions, and describes the properties of reduced ordered decision graphs that make them useful for the manipulation of discrete functions, in general, and for this task, in particular. Section 3 describes how the minimum description length principle is applied and gives the details of the encoding scheme used, which follows closely the approach followed by Quinlan and Rivest (1989). Section 4 contains the central contribution of this paper and describes a simple but effective local optimization algorithm that derives reduced ordered decision graphs of small complexity from a training set. The algorithm proposed initializes the decision graph using one of the techniques described in Section 4.1 and then applies incremental changes to the decision graph that reduce the overall description complexity. Section 5 describes the results of a series of experiments performed with this and other algorithms. The performance of the algorithm is compared with the performance of alternative approaches in a variety of problems defined over discrete and continuous domains. The set of problems analyzed includes problems generated from artificial and real-world domains, as well as a benchmark set assembled by an independent group for the purpose of comparing induction algorithms. Section 6 summarizes the results obtained and proposes some directions for future work in this area.

2. Decision Trees and Decision Graphs

We address the problem of inferring a classification rule from a labeled set of instances, the training set. In particular, we are interested in supervised, single-concept learning in discrete spaces. Let the input space be defined by a set of discretely valued attributes, $D = X_1 \times X_2 \ldots \times X_N$, and let a concept be a subset of the input space. Each instance $(d_i, l_i) \in D \times \{0,1\}$ is described by a collection of discretely valued attributes d_i that defines a point in D and a label l_i that is 1 iff this point belongs to the target concept, C. The training set is a set of instances $T = \{(d_1, l_1), \ldots, (d_m, l_m)\}$. The values $l_1, \ldots, l_m$ define a Boolean vector l with m components that has a 1 in the ith position iff d_i is in C. Boolean vectors will also be used in other contexts and will be manipulated using the standard Boolean operations in the natural way. When describing expressions that involve either Boolean functions or Boolean vectors, we will follow the accepted conventions from the logic synthesis literature. Specifically, we will omit the conjunction operator, use the $+$ symbol to represent disjunction and the $\oplus$ symbol to represent the exclusive-or operator. The norm of a Boolean vector z, $|z|$, represents the number of ones in z.

The objective is to infer a classification rule, or hypothesis, that can be viewed as a function $f : X_1 \times X_2 \times \ldots \times X_N \rightarrow \{0,1\}$ that approximates the characteristic function of concept C.

We are particularly concerned with two representations for functions defined over discrete domains: decision trees and decision graphs. A decision tree is a rooted tree where each non-terminal node is labeled with the index of the attribute tested at that node. From each non-terminal node labeled with variable x_i, $|X_i|$ arcs point to other nodes. Each one of these arcs is labeled with one of the possible values of the attribute x_i. In the case of single concept learning, there are two types of terminal (or leaf) nodes: type 0 and type 1, denoting, respectively, non-inclusion and inclusion in the target concept.

Similarly, a decision graph is a directed acyclic graph where each non-terminal node is labeled with the index of the attribute tested at that node. A decision graph is similar to a decision tree except that the underlying graph may have re-convergent paths. Decision graphs are commonly described as *Boolean decision diagrams* in the logic synthesis literature and as *branching programs* in computer science theory work (Meinel, 1989). Figure 1 shows a decision tree and a decision graph for the function $f : \{0,1\}^4 \rightarrow \{0,1\}$ defined by $f = x_1 x_2 + x_3 x_4$. In this and other figures illustrating functions defined over Boolean spaces, we will use the convention that the rightmost edge leaving each node will always be the one labeled with the value 1.

Decision graphs and decision trees can be used to classify any instance by tracing a path from the root to a terminal node that follows the edges labeled with the values of the attributes present in that instance.

A decision graph is called *ordered* if there is an ordering of the variables such that, for all possible paths in the graph, the variables are always tested in that order (possibly skipping some of them). A decision graph is called *reduced* if no two nodes exist that branch exactly in the same way, and it is never the case that all outgoing edges of a given node terminate in the same node (Bryant, 1986). A graph that is both reduced and ordered is called a *reduced ordered decision graph (RODG)*. For a given ordering of the variables, reduced ordered

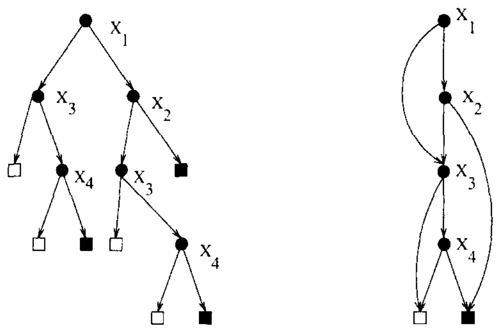


Figure 1. Decision tree for $f = x_1x_2 + x_3x_4$ and the corresponding decision graph.

decision graphs are canonical representations of Boolean functions, a characteristic that makes them specially useful for the manipulation of this type of functions.

In this work, only problems defined over Boolean spaces are considered. Problems defined by multi-valued attributes can be easily mapped into a Boolean space by replacing each multi-valued attribute $x_i \in X_i$ by a set of $\lceil \log_2 |X_i| \rceil$ Boolean-valued attributes. We will henceforth assume that all attributes are Boolean valued and that $D = \{0, 1\}^N = B^N$. The encoding of multi-valued variables as a set of Boolean variables does not change the class of concepts that can be represented in polynomial size by either decision trees of decision graphs, but may still have a considerable impact on the quality of the generalization performed. In most cases, this transformation makes concepts slightly harder to learn because they are represented by more complex trees in terms of the Boolean attributes. In a few other cases, it actually helps the induction process by creating regularities in the data that are not apparent before the transformation. For example, inferring the concept of a prime number is possible if the number is represented in binary but infeasible if the number is represented as an integer-valued attribute. Extending the system to make it able to handle multi-valued attributes directly is an important direction for future research and does not require any fundamental change in the algorithms. It does, however, require the use of a distinct package for the manipulation of discrete functions than the one used.

The present system internally transforms discretely valued attributes into Boolean-valued ones in a way that is transparent to the user. However, if the final RODG needs to be easily understandable (instead of simply used to classify new instances), it is harder to interpret because of the binary coding of multi-valued variables. A post-processing step can be used to recover an RODG formulated in terms of the original variables whenever clarity of the derived rule is a critical factor.

Consider now a decision tree or a decision graph defined over the domain D. Let $n_1, n_2, ..., n_r$ be the nodes in the tree or graph. Let v_i denote the variable tested in node n_i, n_{t_i} (the *then* node) denote the node pointed to by the arc leaving node n_i when v_i is 1 and n_{e_i} (the *else* node) denote the node pointed to by the arc leaving node n_i when v_i is 0. Finally let node n_s be the root of the decision tree or graph. Each node n_i defines a Boolean function $f(n_i) : D \rightarrow B$ defined, in a recursive way, by

$$f(n_i) = \begin{cases} f(n_{t_i}) & \text{if } v_i = 1 \\ f(n_{e_i}) & \text{if } v_i = 0 \end{cases} \tag{1}$$

The recursion stops when n_{e_i} (or n_{t_i}) is a terminal node. In this case $f(n_{e_i})$ (or $f(n_{t_i})$) equals the constant 0 function or the constant 1 function, depending on the type of the terminal node.

We will use y_i to denote the Boolean vector with m components that has a 1 in position j iff the function defined by node n_i has the value 1 for the value of the attributes defined by the jth instance in the training set:

$$y_i^j = f_i(d_j) \tag{2}$$

In an analogous way, V_i will denote the Boolean vector with m components that has a 1 in position j iff the variable v_i has the value 1 for the jth instance in the training set:

$$V_i^j = v_i(d_j) \tag{3}$$

2.1. Manipulating Discrete Functions Using RODGs

For a given ordering of the variables, reduced ordered decision graphs are a canonical representation for functions in that domain (Bryant, 1986). This means that given a function $f : X_1 \times X_2 ... \times X_N \rightarrow \{0, 1\}$ and an ordering of the variables, there is one and only one representation for a function f.

Packages that manipulate RODG's are widely available and have become the most commonly used tool for discrete-function manipulation in the logic synthesis community (Brayton et al., 1990). Some of these packages are restricted to Boolean functions (Brace et al., 1989). In this case, each non-terminal node has exactly two outgoing edges. Other packages (Kam & Brayton, 1990) can accept multi-valued attributes directly, thereby allowing each non-terminal node to have an arbitrary number of outgoing edges.

All these packages provide at least the same basic functionality: the ability to combine functions using basic Boolean and arithmetic operations and the ability to test for containment or equivalence of two functions. They also provide an array of more complex primitives for function manipulation that are not relevant for the work presented here.

Several functions can be represented using a multi-rooted (or shared) RODG and each function is usually represented by a pointer to the RODG node that represents the function. Because RODG's are canonical, the equivalence test (and, therefore, the tautology test) can be performed in constant time. This means that the task of checking for the equivalence of two functions represented by their shared RODGs is a trivial one because it reduces to the comparison of two pointers.[1]

The algorithms described in this paper make use of only a small fraction of the facilities provided by RODG packages. In particular they will only use the following primitives for Boolean-function manipulation:

- Boolean combination of two existing functions. For example, $f := gh$ returns a function f that is the Boolean *and* of two existing functions, g and h.

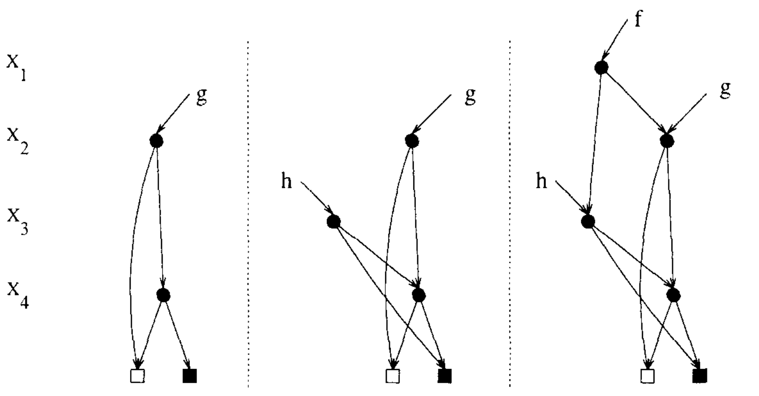


Figure 2. RODGs for the functions $g = x_2x_4$, $h = \overline{x_3} + x_4$ and $f = x_1x_2x_4 + \overline{x_1}(\overline{x_3} + x_4)$.

- Complement of an existing function. Example: $f := \overline{g}$.
- Creation of a function from an existing variable. For example, $f :=$ Fvar(i) returns a function f that is 1 when variable x_i is 1 and is 0 otherwise.
- The *if-then-else* operator. For example, $f :=$ Ite(v, g, h) returns the function g for the points where function v is 1 and the function h for the points where v is 0. Although the *Ite* operator is simply a shorthand for the combination $f := (vg) + (\overline{v}h)$, it has a fundamental role in the definition of RODG operations and deserves separate treatment.

A complete description of the algorithms used by the RODG packages to manipulate Boolean functions is outside the scope of this paper, but we include, in Appendix A a description of the basic techniques designed for this purpose. A more complete exposition can be found in the literature (Bryant, 1986; Brace et al., 1989). For the purposes of understanding the approach outlined in this paper, it is sufficient to understand how the facilities provided by these packages can be used to manipulate Boolean functions.

Example: The function $f : \{0,1\}^4 \to \{0,1\}$ defined in expression (4) can be obtained using the primitives provided by the package. Figure 2 shows the successive RODG's created by the package to represent the functions g, h and f. In these diagrams, a white square represents the terminal node that corresponds to 0, a filled square represents the terminal node that corresponds to 1.

$$f(x_1, x_2, x_3, x_4) = \begin{cases} x_2x_4 & \text{if } x_1 = 1 \\ \overline{x_3} + x_4 & \text{if } x_1 = 0 \end{cases} \tag{4}$$

□

3. Minimizing Message Length and Encoding of RODGs

The tradeoff between hypothesis simplicity and accuracy in the training data is controlled using the minimum description length (MDL) principle of Rissanen (1978,1986). This very general principle can be derived from algorithmic complexity theory in a simple and elegant way (Li & Vitányi, 1994). It states, in a simplified way, that the hypothesis that minimizes the *sum of the length of the hypothesis description with the length of the data encoded using this hypothesis as a way to predict the labels* is the one more likely to be accurate in unseen instances.[2]

Assume that an encoding scheme for describing both RODGs and a list of exceptions has been agreed upon in advance. Let d_g be the description length of an RODG and d_d be the length of the message required to describe the exceptions to this RODG in a given training set. According to the MDL principle, the RODG that minimizes the total description length, $d_g + d_d$, will exhibit the best generalization accuracy.[3]

The computation of d_g, the description length of the RODG, is performed by evaluating the length of a particular encoding. We encode a reduced ordered decision graph using a scheme inspired by the one proposed by Quinlan and Rivest (1989), modified to take into account the fact that a node can be visited more than once. It is also restricted to consider only decision graphs with two terminal nodes, the representation used in this work. Nodes in an RODG are encoded as follows:

- A node that was never visited before is encoded starting with 1 followed by the encoding of the variable tested at that node, followed by the encoding of the node pointed to by the *else* edge, followed by an encoding of the node pointed to by the *then* edge.
- A node that was visited before is encoded starting with 0 followed by a reference to the already described node.

The first node to be described is the root of the graph, and the two terminal nodes are considered visited from the beginning and assigned references 0 and 1. We ignore the issues related with the use of non-integral numbers of bits and we make the description less redundant by noting that when one is deeper in the decision graph not all variables can be usefully tested (only the ones that were not tested previously). Furthermore, when a reference to an already described node is given, only $\log_2(r')$ bits are required, where r' is the number of nodes described up to that point.

We now need to compute d_d, the description length of the exceptions to a given RODG present in the training set. Exceptions to the RODG will be encoded as a string of 0's and 1's where the 1's indicate the locations of the exceptions. In general, the strings have many more 0's than 1's. Again we follow closely the encoding used by Quinlan and Rivest. Assume that there are k 1's in the string and the strings are of length m, known in advance.

We can encode the string by first sending the value of k, which requires $\log_2(m)$ bits and then describing which string with k 1's we are referring to. Since there are $\binom{m}{k}$ such strings we find that

$$d_d = \log_2(m) + \log_2 \binom{m}{k} \tag{5}$$

Using Stirling's formula to approximate the second term in (5) we obtain

$$d_d = mH\left(\frac{k}{m}\right) + \frac{3\log_2(m)}{2} - \frac{\log_2(k)}{2} - \frac{\log_2(m-k)}{2} - \frac{\log_2(2\pi)}{2} + O\left(\frac{1}{m}\right) \tag{6}$$

where $H(p)$ is the usual entropy function

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p) \tag{7}$$

Wallace and Patrick (1993) point out that the coding scheme proposed by Quinlan and Rivest is sub-optimal for some types of trees. In particular, they analyze the deficiencies of this coding scheme for non-binary trees and for trees with more than two classes. Their analysis, however, does not hold for the decision-graph case since it makes use of specific characteristics of trees that are not shared by graphs. Our encoding scheme may, however, suffer from other inefficiencies because it does not make use of all the prior knowledge about graph structure. In particular, it is known that nodes that are lower in the decision graph will be reused and therefore referenced more often than nodes at upper levels. Conceptually, this could be used to our advantage in building a more efficient coding scheme by assigning shorter codes to nodes closer to the terminal nodes. In practice, it is hard to use this knowledge to achieve significant gains in coding length because little is known about the distributions of references to nodes in typical problems.

4. Deriving an RODG of Minimal Complexity

This section contains the central contribution of this paper and describes in detail the algorithms used to derive an RODG of minimal complexity. The RODG that serves as the starting point for the local optimization algorithm, described in Section 4.2, is obtained from the training set data using one of the techniques described in Section 4.1. Section 4.3 describes the algorithms that select the best ordering of the variables.

4.1. Generating the Initial RODG

There are several possible ways to generate an RODG that can be used as the starting point for the local optimization algorithm. Our experiments have shown that three of them are particularly effective, although the relative size of the RODGs generated by different methods varies strongly from problem to problem (Oliveira, 1994). The RODG selected as the initial solution is the RODG that exhibits the smallest total description length of the following three candidates:

- The RODG that realizes the function implemented by a decision tree derived from the training set data using a standard decision-tree algorithms.
- The RODG that realizes the function implemented by a decision tree defined over a new set of variables obtained using constructive-induction techniques.

- The RODG obtained by applying the *restrict* heuristic (Coudert et al., 1989) to the function obtained by listing all positive instances in the training set.

We now describe in more detail how each one of these techniques can be used to obtain an RODG that serves as the starting point for the local optimization algorithm.

4.1.1. Initialization Using Decision Trees

One way to initialize the RODG is to obtain a decision tree from the data and to convert the function obtained by the decision tree to RODG form.

Several efficient algorithms for the induction of decision trees from data have been proposed in the literature. Since the attributes are Boolean and we are not concerned with algorithms for pruning the tree, we can use a relatively straightforward algorithm to generate the decision tree. A simplified version of ID3 (Quinlan, 1986) is used to generate the decision tree. The decision tree is built in a recursive way, by selecting, at each point, the attribute to be tested as the one that provides the larger amount of information about the class of the instances that reached that node.

Example: Figure 1, used as the example in Section 2 shows a decision tree for the function $f = x_1x_2 + x_3x_4$ and the decision graph that results from the application of this technique, assuming the ordering used is (x_1, x_2, x_3, x_4). □

4.1.2. Initialization Using a Constructive Induction Algorithm

Constructive induction algorithms create new complex attributes by combining existing attributes in ways that make the description of the concept easier. The *fulfringe* constructive induction algorithm (Oliveira & Vincentelli, 1993) identifies patterns near the fringes of the decision tree and uses them to build new attributes. The idea was first proposed by Pagallo and Haussler (1990) and further developed by other authors. A constructive induction algorithm of this family, *dcfringe* (Yang et al., 1991) identifies the patterns shown in the first two rows of Figure 3. *Fulfringe* identifies all the patterns used by *dcfringe* but also identifies additional ones that correspond to functions poorly correlated with the input variables. These additional patterns are listed on the third row of Figure 3. In this figure nodes not marked with squares can be either terminal or non-terminal nodes. This means that the patterns in the second row are more specific than the patterns of the first row.

Note that the creation of a function or its complement is equivalent, from a constructive induction point of view. The constructive induction algorithm *dcfringe* can be reformulated in such a way that only conjunctions of existing attributes are created. Similarly, the patterns in the last row of Figure 3 can all be associated with the exclusive-or function of existing attributes. We opted, however, for a formulation closer to the one that was used by other authors in this field.

The new composite attributes are added (if they have not yet been generated) to the list of existing attributes and a new decision tree is built. The process is iterated until no further

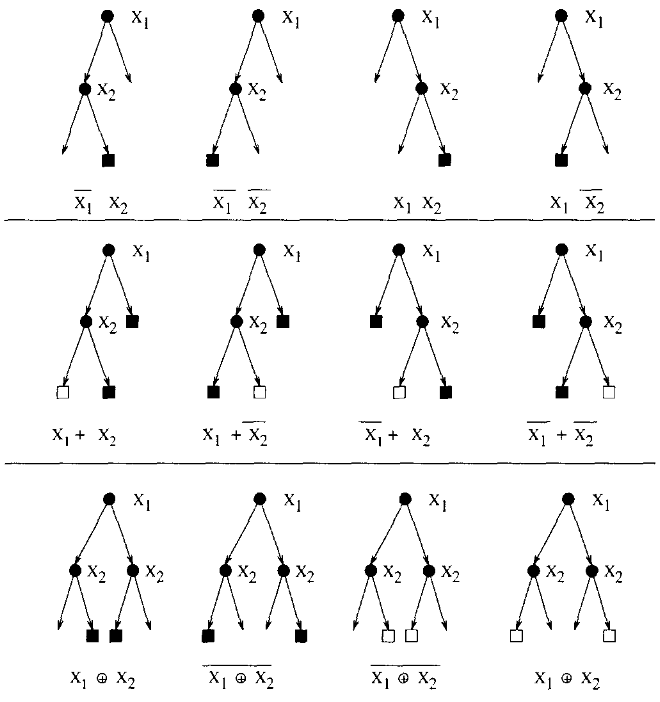


Figure 3. Fringe patterns identified by *fulfringe*. Each partial subtree shown corresponds to the Boolean function of two variables described below it. Nodes without squares can be either terminal or non-terminal.

reduction in the decision-tree size takes place or a decision tree with only one decision node is built.[4]

Since the composite attributes are Boolean combinations of existing attributes, the RODGs for them are created in a straightforward way using the Boolean operations between existing functions provided by the RODG package. Expression (1) can still be used to derive the RODG implemented by a decision tree defined over this extended set of variables, but the variable v_i will not refer, in general, to a primitive attribute. This is handled in a transparent way by the functions available in the RODG package, as described in Appendix A.

Note that even though the successive decision trees are defined using composite attributes, the RODGs that correspond to any one of these trees are still defined over the original set of variables. In this way, the constructive induction algorithm is used only to derive a simpler Boolean function to initialize the RODG reduction algorithm, not to add new variables as in standard constructive induction approaches.

Example: Figure 4 shows the successive decision trees obtained using this algorithm for the function used in the previous example. The first decision tree created is the same as before. Using the patterns listed in Figure 3 the algorithm creates the two following attributes: $x_5 = x_1x_2$ and $x_6 = x_3x_4$. RODGs for these new attributes are also created, as they will be needed in the next step.

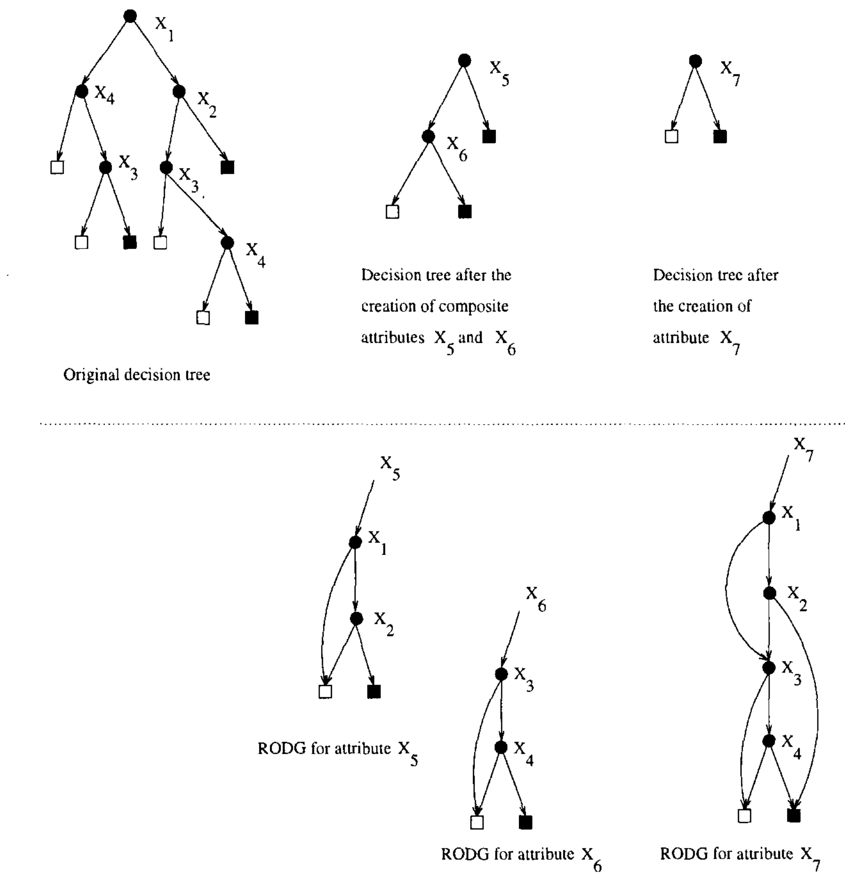


Figure 4. In this example, the composite attributes $x_5 = x_1 x_2$, $x_6 = x_3 x_4$ and $x_7 = x_5 + x_6$ are created by the constructive induction algorithm. The top half of the figure shows the successive decision trees created by *fulfringe* while the bottom half shows the decision graphs created for each one of the newly created attributes. The decision graph for x_7 is the one returned by the procedure.

A smaller decision tree is then built using these attributes (together with the primitive ones, in general) and the new attribute $x_7 = x_5 + x_6$ is created, as well as the RODG for x_7 as a function of $(x_1 \ldots x_4)$. The RODG created for the new composite attribute x_7 is the same as the RODG for the final function, because the last decision tree created has only one node. In this case, the final RODG is the same as the one obtained using the initial decision tree although, in general, this is not the case. $\square$

4.1.3. Initialization Using the Restrict Operator

The third way to initialize the algorithm is to use algorithms for RODG reduction like the restrict operator (Coudert et al., 1989). This RODG operator can be used to obtain a more compact RODG representation for an incompletely specified function.

The restrict operator belongs to a family of heuristics (Shiple et al., 1994) that generate a small RODG by merging, in a bottom-up fashion, nodes in an RODG. The merging of nodes is performed in a way that keeps the RODG consistent with the training set data.

Two RODGs are required to apply the restrict heuristic: an RODG that describes the function f to be *restricted* and an RODG that describes the *care set*, i.e., the points in the input space where the value of the function is relevant. The first RODG is created by considering a function that is 1 for all positive instances and 0 otherwise. The care set consists off all the points in the input space that are present in the training set, either as positive or negative instances of the target concept[5]. The restrict heuristic is then applied to obtain a small RODG that is consistent with the training set, but is, in general, much smaller than the RODG for the original function f.

Although a full description of this algorithm is outside the scope of this paper, we will use Figure 5 to provide a simple illustration on how the procedure works. In that figure, terminal nodes marked with a cross are points in the input space that do not belong to the *care* set. The value of the function for these points can therefore be chosen as to minimize the size of the resulting *RODG*. The algorithm works as follows: starting at the bottom, each node is examined to check if its children can be merged. In this example, the algorithm verifies that it can merge, in a pairwise manner, the children of nodes 5, 6 and 7. In the second step, it verifies that the children of node 3 can also be merged. Since no more children can be merged, the algorithm stops and returns the last RODG shown as the result.

It is clear that by deciding prematurely which nodes can be merged, the algorithm can make the wrong choices, as happens in this example. Variations on this method and a complete analysis of alternative methods to chose mergings are analyzed in depth by Shiple (1994).

The restrict heuristic is remarkably fast and obtains, in some cases, RODGs that are much better solutions than the ones obtained by the much slower decision tree algorithms. However, in problems that have many attributes and where the positive and negative instances can be separated by a small subset of the available attributes this heuristic tends to generate RODGs that depend only on a small subset of the attributes, namely the ones that come first in the ordering selected (Oliveira, 1994).

4.2. Reducing an RODG

After creating the initial RODG using one of the methods described above, a local optimization algorithm will be used to reduce its size. The search for an RODG of minimum total description length is performed in steps. Each step decreases the description length of the RODG, possibly at the expense of decreased accuracy in the training set. A step is accepted if the total description length after applying that step is less than the previous description length.

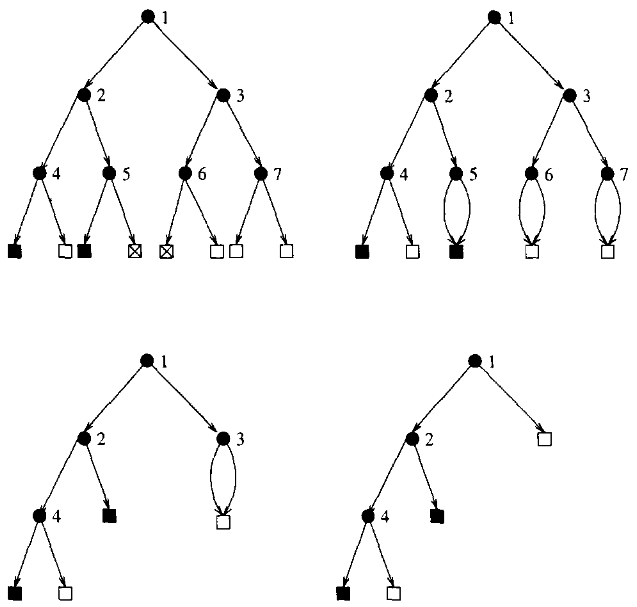


Figure 5. The restrict heuristic illustrated. At each step, the *restrict* heuristic examines if the two children of a given node can be merged without altering the value of the function for any point in the *care* set.

At each step, one or more nodes are removed from the RODG. We associate with each node n_i in the RODG a Boolean vector w_i that is 1 in position j iff the jth instance defines a path in the RODG that goes through node n_i. The jth position of vector w_i is denoted by w_i^j. Since all instances need to go through the root node, n_s, $w_s^j = 1$ for all j. The remaining w_i vectors can be computed applying recursively the following Boolean expressions:

$$w_i = \Big(\sum_{j \text{ s.t. } n_{e_j} = n_i} \overline{V_j} w_j \Big) + \Big(\sum_{j \text{ s.t. } n_{t_j} = n_i} V_j w_j \Big) \tag{8}$$

The Boolean vectors w_i define the set of instances that have to be taken into consideration if the function of node n_i is to be changed. Changes to node n_i will only affect instances that have the corresponding bit set in vector w_i.

4.2.1. Removing One Node by Redirecting Incoming Edges

The *RemoveNode* procedure reduces the description length by making one of the nodes in the RODG redundant. This is done by redirecting all its incoming edges. When node n_i is under consideration, the algorithm goes through all incoming edges and selects, for each one of them, a different node n_k that corresponds to a function as close to the target as possible (see Figure 6).

The value of this function is only important for the instances that reach n_i through the edge that is being redirected. The pseudo-code in Table 1 describes how this modification

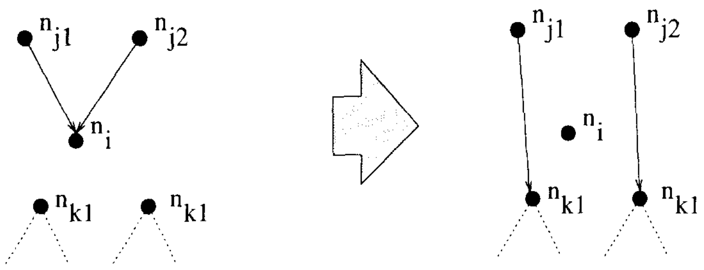


Figure 6. Removing one node from the RODG.

Table 1. Pseudo-code for the procedure that removes a node from the RODG.

```
RemoveNode(R)
    foreach $n_i$
        foreach $j$ s.t. $n_{e_j} = n_i$                      For nodes that have the else edge pointing to $n_i$
            $w := w_j \overline{V_j}$                          Instances reaching $n_i$ through this edge
            Select $n_k$ such that $|(y_k \oplus l)w|$ is minimal    Find a node with a function as similar
                                                               as possible for the relevant instances
            Modify RODG such that $n_{e_j} = n_k$
        foreach $j$ s.t. $n_{t_j} = n_i$                      For nodes that have the then edge pointing to $n_i$
            $w := w_j V_j$                                     Instances reaching $n_i$ through this edge
            Select $n_k$ such that $|(y_k \oplus l)w|$ is minimal    Find a node with a function as similar
                                                               as possible for the relevant instances
            Modify RODG such that $n_{t_j} = n_k$
        if Modified RODG has a smaller description length
            return Modified RODG
        else
            Undo changes
    return Failure
```

is accomplished. This algorithm takes as input one copy of the current RODG and tries, for each node, to redirect its incoming edges. If, for some node, the RODG that results from redirecting each one of these edges has a total description length smaller than the original one, the procedure returns the modified RODG.[6]

4.2.2. *Replacing a Pair of Nodes by a New Node*

If the procedure *RemoveNode* fails to make one node redundant, the more expensive procedure *ReplacePair* is called. *ReplacePair* removes a pair of nodes by creating a new node that implements a function as close as possible to the functions implemented by the pair of nodes under consideration (see Figure 7). The value of the new function is only relevant for

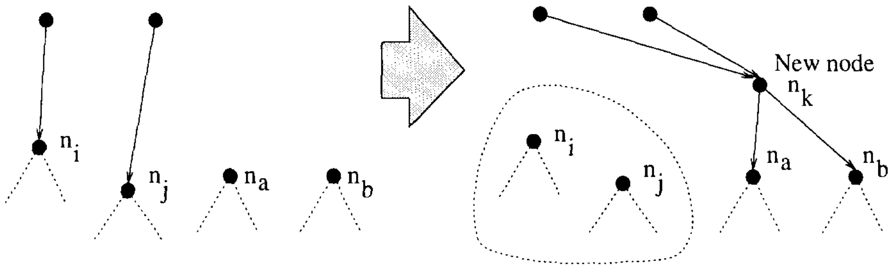


Figure 7. Replacing a pair of nodes by a new node.

the instances that reach the nodes being considered for removal. The pseudo-code in Table 2 describes how the new node is selected, by creating a new function that differs from the functions implemented by the nodes under consideration for as few instances in the training set as possible.[7]

Table 2. Pseudo-code for the procedure that replaces a pair of existing nodes by a newly created node.

```
ReplacePair(R)
    **foreach** $n_i$
        **foreach** $n_j$                                                   *For each pair of nodes*
            $w := w_i + w_j$                     *w is 1 for all instances that reach nodes $n_i$ or $n_j$*
            Create $n_k = \overline{v_k} f(n_a) + v_k f(n_b)$ such that $|(y_k \oplus l)w|$ is minimal
            Modify RODG such that incoming edges into $n_i$ and $n_j$ point to $n_k$
            **if** Modified RODG has smaller description length
                **return** Modified RODG
            **else**
                Undo changes
    **return** Failure
```

4.3. Selecting the Best Ordering

The selection of a good ordering for the variables is of critical importance if the goal is to obtain a compact RODG. Regrettably, selecting the optimal ordering for a given function is NP-complete (Tani et al., 1993) and cannot be solved exactly in most cases. For this reason, and because it is a problem of high practical interest in logic synthesis, many heuristic algorithms have been proposed for this problem (Friedman & Supowit, 1990).

In our setting, the problem is even more complex because we wish to select the ordering that minimizes the final RODG and this ordering may not be the same as the one that minimizes the RODG obtained after the initialization step. Our implementation uses the *sift* algorithm (Rudell, 1993) for dynamic RODG ordering. It has been observed by a number of different authors (Brace et al., 1989; Ishiura et al., 1991) that swapping the order of two adjacent variables in the RODG ordering can be done very efficiently because only the nodes in these two levels are affected. The *sift* algorithm selects the best position in the ordering for a given variable by moving that variable up and down (using the inexpensive swap operation) and recording the smaller size observed. This procedure is applied once to all variables and can be, optionally, iterated to convergence. This algorithm is extremely efficient since it was designed to be applied to very large RODGs. Therefore, it can be applied after each step in the RODG reduction algorithm.

4.4. Implementation Considerations

The complexity of these algorithms depends strongly on the approach used to evaluate the description-length reduction achieved by each operation. Because the effect of each change can be estimated locally, recomputing the overall description length of the RODG or the number of exceptions created by a local modification is not required.

With careful coding, the *RemoveNode* procedure requires $O(r^2m)$ operations, where, as before, r is the number of nodes in the current RODG and m is the size of the training set. The *ReplacePair* procedure is more expensive and requires $O(r^3m)$ operations. By using bit-packing techniques the algorithm can be used to reduce RODGs with hundreds or a few thousands of nodes.

For very large problems the decision graph obtained from the initialization phase may be too large (Oliveira, 1994). In this case, the local optimization algorithm may take a long time to reduce this graph. For these problems, the algorithm can be run in a fast mode that initializes the graph with a decision tree that is not fully consistent with the training set data. This is obtained by stopping the growth of the decision tree when the entropy of the samples that reach a particular node is less than a given value. The larger this value, the smaller the decision tree obtained and the simpler the initial decision graph. However, if this threshold is set too high, the local optimization algorithm will not be able to improve the solution and the generalization accuracy obtained by the decision graph will not be any better than the one obtained by the decision tree that was used in the initialization.

The algorithms described in Sections 4.1 through 4.3 are combined in a straightforward way as the pseudo-code in Table 3 shows.

These algorithms were implemented in a system called *smog (Selection of Minimal Ordered Graphs)* that uses the CMU RODG package (Brace et al., 1989) to perform the standard RODG manipulations.

5. Experiments

To evaluate the effectiveness of the approach presented here, we tested the algorithm on a set of problems that have been addressed previously in the machine learning literature and that are either widely available or easily reproducible. We also present a brief summary of a comparison performed using another benchmark developed by an independent group for the purpose of comparing learning algorithms in problems that require complex descriptions.

Table 3. The main loop of the *smog* algorithm.

```
MainLoop()
    $S$ := InitRodg()
    **repeat**
        $R := S$                                   *Store the current RODG*
        $R$ := Reorder($R$)                        *Select best ordering for current RODG*
        $S$ := RemoveNode($R$)
        **if** $S$ = Failure                       *RemoveNode operation failed*
            $S$ := ReplacePair(R)
    **until** S = Failure
    **return** R
```

5.1. Results on Problems From the Literature

In this section, the comparison between different algorithms was made using three sets of problems with distinct origins: the set of 8 problems proposed by Pagallo and Haussler (1990), a set of 5 problems selected because they are known to accept small decision graph solutions but require comparably larger decision tree representations (Oliveira, 1994), and a set of 13 problems from the U.C. Irvine repository (Murphy & Aha, 1991).

5.1.1. Experimental Setup

For each of the problems selected, we compared the performance of *C4.5*, a popular decision tree induction algorithm (Quinlan, 1993), with *smog*, the system that implements the algorithms described in Section 4. The comparison was performed using the commonly accepted *10-fold cross-validation* methodology and the statistical significance of the results was evaluated using a one-tailed paired t-test (Casella & Berger, 1990). All problems with continuously valued attributes were discretized using the entropy-based method of Fayyad and Irani (1993). Recent results (Dougherty et al., 1995) have shown that the performance of induction algorithms and, in particular, of C4.5, does not degrade and may even improve if this discretization method is applied to problems with continuously valued attributes. Multi-valued attributes, either originally present in the problem or obtained after the discretization step, were encoded using the binary encoding method described in Section 2.

Table 4 lists the average generalization error for the *C4.5* and *smog* algorithms on the set of problems selected.

Typical learning curves for some of these problems are shown in figure 8. These curves were obtained by setting aside 20% of the data for the test set and generating increasing larger training sets. Each curve represents the average of 10 runs performed using this methodology. These curves show that the differences in accuracy observed suffer great variations from problem to problem. In some cases, an accurate hypothesis is discovered much more rapidly with decision graphs than with decision trees, leading to strongly distinct

Table 4. Average errors for *C4.5* and *smog*. A circle in a given row marks the algorithm that obtained the lowest average error in the given problem. A filled circle means that the difference observed is statistically significant at a confidence level of 95%. The three groups of problems shown correspond to three different sets of concepts proposed in the machine learning literature (Pagallo & Haussler, 1990; Oliveira, 1994; Murphy & Aha, 1991).

Problem	Dataset size	Average error	
		smog	C4.5
dnf1	1000	○ 18.10 ± 7.26	21.00 ± 3.77
dnf2	1000	● 2.70 ± 1.25	12.20 ± 3.19
dnf3	1000	● 1.90 ± 1.10	7.30 ± 3.83
dnf4	1000	● 6.60 ± 2.27	31.10 ± 4.61
mux6	200	○ 0.00 ± 0.00	1.50 ± 4.74
mux11	1000	● 0.00 ± 0.00	5.80 ± 5.09
par4_16	1000	● 0.00 ± 0.00	18.30 ± 12.27
par5_32	1000	● 0.00 ± 0.00	50.50 ± 5.54
kkp	2000	● 0.00 ± 0.00	2.00 ± 1.39
heel	1000	● 0.00 ± 0.00	22.40 ± 5.70
heel9	1000	● 0.00 ± 0.00	1.00 ± 1.05
sm12	1000	● 0.00 ± 0.00	4.80 ± 1.69
str18	1000	● 0.30 ± 0.67	9.10 ± 2.73
krkp	3196	○ 0.31 ± 0.26	0.52 ± 0.45
monk1	432	0.00 ± 0.00	0.00 ± 0.00
monk2	432	● 0.00 ± 0.00	32.83 ± 10.66
monk3	432	0.00 ± 0.00	0.00 ± 0.00
vote	435	5.29 ± 2.64	○ 4.63 ± 3.05
tictactoe	958	● 2.82 ± 1.97	7.07 ± 1.82
breast	699	6.72 ± 2.44	○ 5.85 ± 3.32
credit	690	19.57 ± 5.08	● 14.03 ± 3.28
ion	351	○ 7.71 ± 4.48	8.57 ± 6.45
diabetes	768	23.56 ± 3.67	○ 22.93 ± 3.98
german	1000	34.40 ± 5.25	● 28.70 ± 6.96
glass	214	○ 7.49 ± 7.10	8.88 ± 6.77
vehicles	846	○ 12.88 ± 4.27	13.24 ± 4.45
heart	270	○ 23.33 ± 9.73	24.05 ± 7.24

learning curves. On the other hand, in the majority of the problems where C4.5 performs better, qualitatively similar learning curves are observed with a roughly constant difference favoring the decision tree algorithm.

5.2. Results on the Wright Laboratory Benchmark Set

The results obtained in this Sections's set of problems were obtained by the Pattern Theory Group at the Air Force Wright Laboratory. This group tested *smog* in a benchmark assembled for the purpose of evaluating the effectiveness of a set of learning algorithms (Goldman, 1994). The reader is referred to this reference for a complete description of

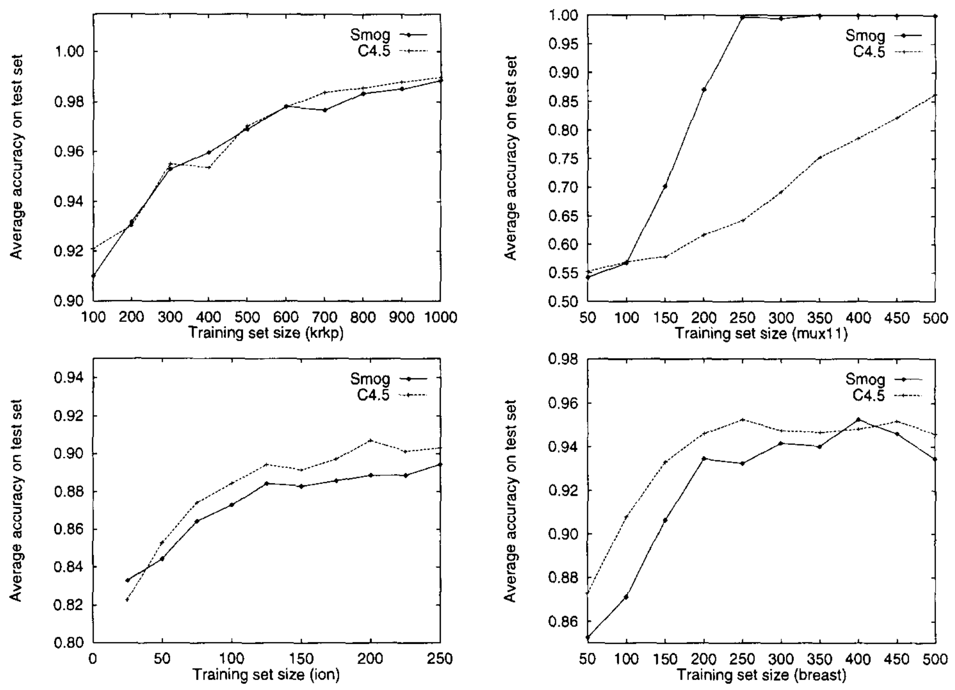


Figure 8. Learning curves for the problems *krkp, mux11, ion* and *breast.*

the methodology adopted and the set of problems addressed. Each one of the problems is defined by a noise-free concept defined over a space of eight Boolean attributes. The majority of the problems represent relatively complex concepts defined over this space. For each of the problems, 10 independent runs with training sets of size 125 were performed and the results were tested in the full dataset for each problem.

The plots in Figure 9 show the compared performance of these two algorithms in graphical form, together with the performance of two alternative approaches for induction:

- Nearest neighbor: an instance is classified as belonging to the same class as the nearest neighbor found in the training set.
- Minimal consistent DNF: the minimum DNF expression consistent with the training set data is found using a two-level logic minimizer (Brayton et al., 1984), *espresso.* This expression is then used to classify unseen instances.

For these problems, the generalization accuracy was tested on a set of instances that also includes the instances used for training. This evaluation methodology puts algorithms like *smog* and *C4.5* at a disadvantage because, unlike *nearest neighbor* or *minimal consistent DNF,* they are not guaranteed to perform perfectly in the training set data.

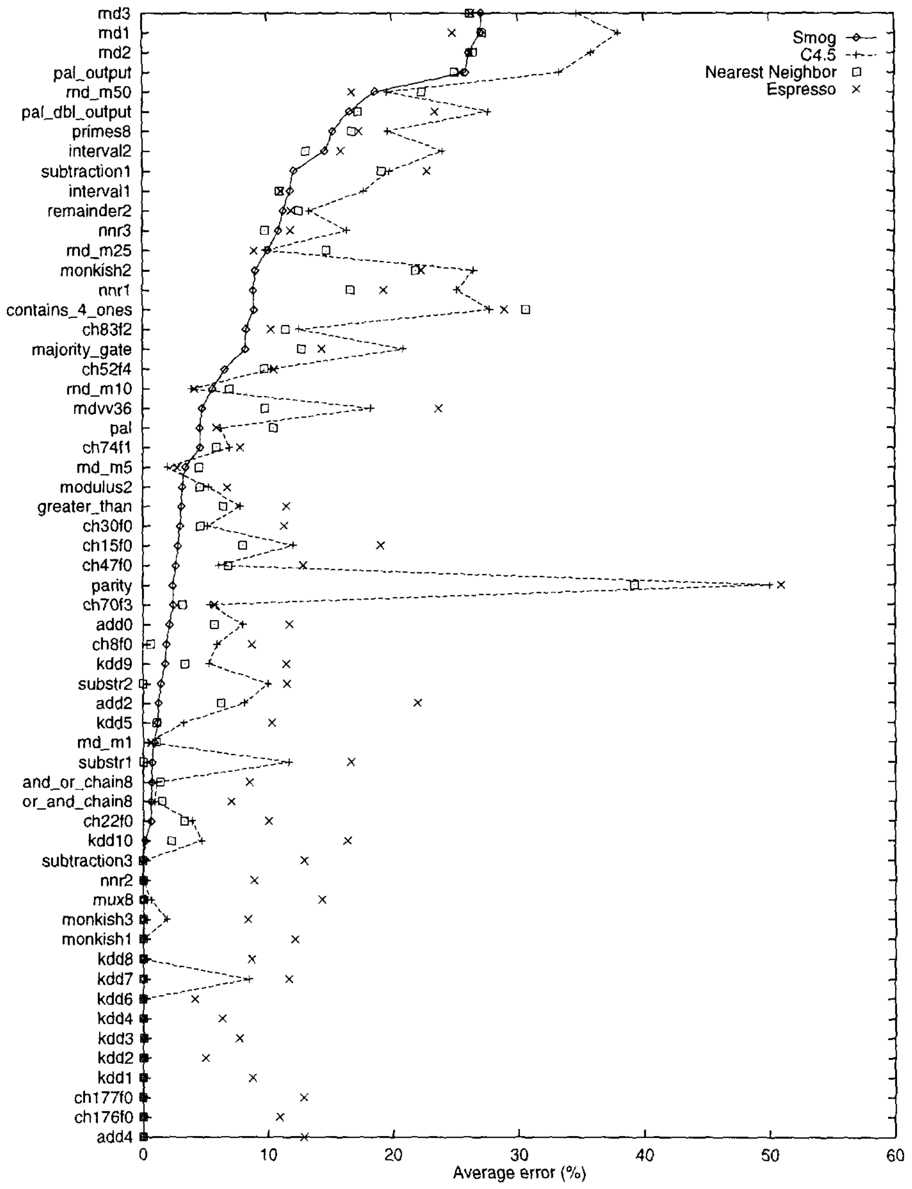


Figure 9. Generalization errors for *smog*, *c4.5*, *espresso* and *nearest neighbor* for the problems in th Wright laboratory benchmark. The concepts are sorted by order of decreasing accuracy for the *smog* algorithm.

5.3. Analysis

The results presented in the two previous sections show that, for problems in the first two groups shown, the algorithm developed for the induction of decision graphs outperforms a commonly used algorithm for the induction of decision trees. For problems in the third group (those taken from the UCI repository), there is no clear advantage from either the decision tree or the decision graph algorithms.

The decision graph approach tends to outperform decision tree algorithms for problems that either exhibit regularities (and therefore require subtree replication) or are highly disjunctive, i.e., are represented by the union of many separate regions of the inputs space. This behavior was observed for all problems that are related with game domains (*tictactoe, krkp, kkp*) and for the majority of problems that are defined by compact Boolean expressions.

The decision graph algorithms presented are also very effective for problems where the selection of the appropriate ordering is important and this ordering can not be easily obtained using the greedy approach commonly adopted by decision tree algorithms. For instance, the *mux11* problem[8] accepts a minimum representation that is, in fact, a tree, but the right ordering is hard to find. In this case, the gain is caused not so much by the use of decision graphs as the underlying representation but by the application of reordering algorithms that are effective in selecting the right ordering. Figure 10 depicts the decision graph obtained from a decision tree after the initialization phase and the final decision graph for one run of the algorithm on the *mux11* problem. This comparison shows the effectiveness of the algorithms for graph reduction, and of the reordering algorithm in this particular case. The reduction of the description size observed in this problem is typical of the reduction observed in the majority of the cases where *smog* outperforms *C4.5* (Oliveira, 1994).

For problems that are defined by sets of continuous attributes, little or no gain was obtained by the use of the decision graphs. This may be due to the fact that these problems are inherently less disjunctive, thereby making little use of the ability of decision graphs to find repeated patterns in the input space or may be due to other limitations of the algorithms.

For the set of problems in the Wright Laboratories benchmark, which require, in the average, more complex descriptions, the decision graph algorithms performed systematically better than the alternatives tested. It is known that many of these concepts are highly disjunctive and require very complex decision surfaces. *Smog* and *C4.5* did better than the two other alternatives, despite the fact that the evaluation methodology was biased against these two algorithms. *Smog* exhibited a highly desired robustness for very disjunctive concepts that tend to disrupt the other algorithms.

In problems that require exponentially large decision trees, the improvements in performance obtained by the use of decision graphs can be radical. However, as the dimension of the problems grows, the high time complexity of the decision graph algorithms makes them less useful. Given the above results, these algorithms seem specially well adapted for the induction of concepts that require relatively involved descriptions and are defined over discrete spaces with a limited number of dimensions.

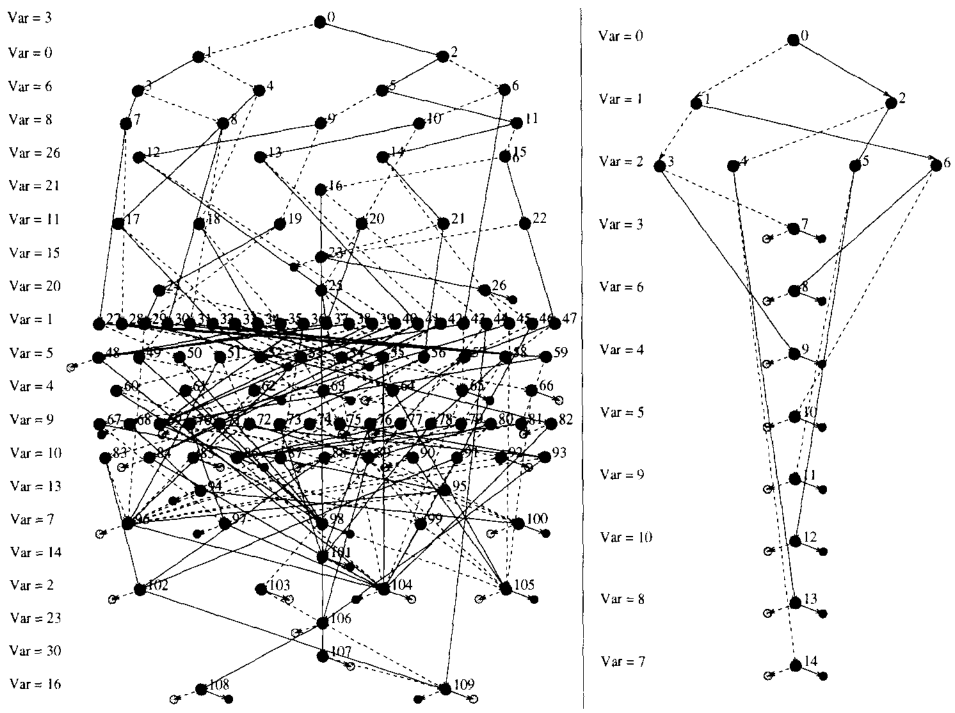


Figure 10. Decision graph obtained after initialization from a decision tree and final decision graph after redundant node removal and reordering was performed.

6. Conclusions and Future Work

We presented an algorithm (*smog*) for the induction of reduced ordered decision graphs from data and evaluated the effectiveness of this approach against alternative approaches. The approach described uses RODGs to represent both intermediate functions and the final hypothesis.

The experimental results have shown that for an interesting class of problems the bias for small RODGs is appropriate and the generalization accuracy observed is better than the one obtained with alternative techniques. Problems that involve highly disjunctive concepts (i.e., concepts represented by many disconnected regions in the input space) stand to benefit most from this approach. For concepts defined by at least partially smooth surfaces over continuous spaces, the approach presented here exhibited little or no gain when compared to standard decision tree algorithms.

The algorithms described in this paper are considerably slower than standard decision tree algorithms. This slowdown may be acceptable in many applications, but intolerable in others. Ultimately, the user has to decide if the increased generalization accuracy is important enough to offset the extra CPU time. We believe this will be the case in many practical applications.

The algorithms manipulate the representation internally using binary RODGs. This makes it necessary to map multi-valued attributes to Boolean valued ones before induction is performed. It is unclear, at this point, how important is the ability to use multi-valued RODGs directly, but this topic deserves further investigation. Multi-category tasks can, in principle, be handled within the current framework. All the algorithms described in this paper can be extended to the case where more than two types of terminal nodes exist. Furthermore, this functionality can be obtained with the currently used RODG package, requiring only relatively minor changes in the algorithms.

Finally, although the algorithm based on incremental modifications outlined in Section 4 is reasonably fast and efficient, it is possible that alternative solutions to this problem can yield even better results. In many cases, the local optimization algorithm did not obtain solutions close to the known optimal solutions and other approaches that are not based on greedy local changes may yield better results. The study of these approaches is left as future work.

In the present version, smog can be used as a direct replacement for popular decision tree algorithms like C4.5, with continuously-valued attributes being discretized internally by the system before the decision graph algorithms are applied. Smog was implemented in C++ and runs in a wide variety of platforms. A copy of the program can be obtained by contacting one of the authors.

Acknowledgments

A shorter version of this paper was presented at the 1995 International Conference on Machine Learning. We are grateful to Timothy Ross and his group at the Air Force Wright Laboratory for making available the experimental results presented in Section 5.2. The authors also wish to thank the editor and the anonymous reviewers for their useful suggestions. Their comments significantly improved the contents and readability of this paper. This work was sponsored in part by the US Joint Services Electronics Program, Contract Number F49620-94-C-0038.

Appendix A

Algorithms for RODG Manipulation

This appendix gives a brief overview of the algorithms that were developed for RODG manipulation and follows closely in form and content the work of Brace (1989). For a much more complete description of the algorithms used, the interested reader should consult this reference. This exposition is uniquely concerned with RODGs defined over Boolean spaces.

Each non-terminal node n_i in the RODG represents a Boolean function denoted by $f(n_i) = v_i f(n_{t_i}) + \overline{v_i} f(n_{e_i}) \equiv (v_i, f(n_{t_i}), f(n_{e_i}))$, where v_i is the variable tested at node n_i, and n_{t_i} and n_{e_i} are the nodes pointed to by the *then* and *else* edges of node n_i, respectively.

The fundamental operation implemented by the RODG package is the *Ite* operator, defined as:

$$\text{Ite}(f, g, h) = fg + \overline{f}h \tag{A.1}$$

It is a simple exercise to verify that all the basic Boolean operations of two variables can be defined using the *Ite* operator with appropriate arguments. For example, $f = ab$ is equivalent to $f = \text{Ite}(a, b, 0)$ and $f = a \oplus b$ is equivalent to $f = \text{Ite}(a, \overline{b}, b)$.

Shannon's decomposition theorem (Shannon, 1938) states that

$$f = vf_v + \overline{v}f_{\overline{v}} \tag{A.2}$$

where v is a variable and f_v and $f_{\overline{v}}$ represent f evaluated at $v = 1$ and $v = 0$, respectively.

Let w be a variable and $f(n_i) = (v_i, f(n_{t_i}), f(n_{e_i}))$ and assume that either w comes before v_i in the ordering or that $v_i = w$. Finding the cofactors of f with respect to w is trivial because, in the first case, f is independent of w:

$$f_w = \begin{cases} f & \text{if } v_i \neq w \\ f(n_{t_i}) & \text{if } v_i = w \end{cases} \qquad f_{\overline{w}} = \begin{cases} f & \text{if } v_i \neq w \\ f(n_{e_i}) & \text{if } v_i = w \end{cases} \tag{A.3}$$

The following recursive definition gives a simple algorithm for the computation of the function $z = \text{Ite}(f, g, h)$. Let v be the top variable of f, g and h. Then,

$$\begin{aligned} z &= vz_v + \overline{v}z_{\overline{v}} \\ &= v(fg + \overline{f}h)_v + \overline{v}(fg + \overline{f}h)_{\overline{v}} \\ &= v(f_v g_v + \overline{f}_v h_v) + \overline{v}(f_{\overline{v}} g_{\overline{v}} + \overline{f}_{\overline{v}} h_{\overline{v}}) \\ &= \text{Ite}(v, \text{Ite}(f_v, g_v, h_v), \text{Ite}(f_{\overline{v}}, g_{\overline{v}}, h_{\overline{v}})) \\ &= (v, \text{Ite}(f_v, g_v, h_v), \text{Ite}(f_{\overline{v}}, g_{\overline{v}}, h_{\overline{v}})) \end{aligned} \tag{A.4}$$

The terminal cases for this recursion are:

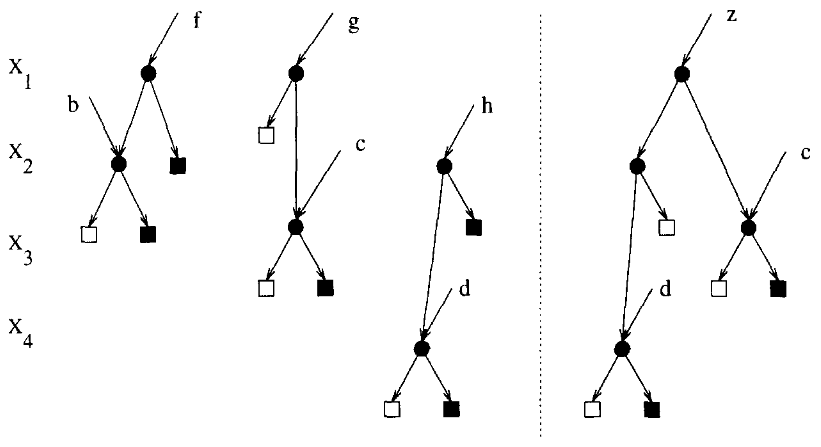
$$\text{Ite}(1, f, g) = \text{Ite}(0, g, f) = \text{Ite}(f, 1, 0) = f \tag{A.5}$$

This procedure would have exponential complexity even for functions with small RODGs if the recursion is used all the way down to the terminal cases in every call of the *Ite* function. This exponential complexity is avoided by keeping a table of existing functions. Each element in the table is a triple (v, g, h) and each node in the RODG corresponds to an entry in this table. Before applying the recursive definition (A.4), the algorithm checks if the desired function already exists.

Example: Figure A.1 shows an example of the application of the recursive definition in the computation of the function $z = \text{Ite}(f, g, h)$, where the functions f, g, and h are shown in the left side of the figure.

For clarity, several copies of the terminal nodes are shown. The reader should keep in mind that only one copy of each function is kept at any time. This is true for the terminal nodes and also for the nodes that implement the functions c and d, but depicting only one copy of these nodes would make the diagram too complex to be useful. In this example, the nodes that correspond to the functions c and d already exist and do not need to be created from scratch.

□



$$\begin{aligned} z &= \mathrm{Ite}(f, g, h) \\ &= (x_1, \mathrm{Ite}(f_{x_1}, g_{x_1}, h_{x_1}), \mathrm{Ite}(f_{\overline{x_1}}, g_{\overline{x_1}}, h_{\overline{x_1}})) \\ &= (x_1, \mathrm{Ite}(1, c, h), \mathrm{Ite}(b, 0, h)) \\ &= (x_1, c, (x_2, \mathrm{Ite}(b_{x_2}, 0_{x_2}, h_{x_2}), \mathrm{Ite}(b_{\overline{x_2}}, 0_{\overline{x_2}}, h_{\overline{x_2}}))) \\ &= (x_1, c, (x_2, \mathrm{Ite}(1, 0, 1), \mathrm{Ite}(0, 0, d))) \\ &= (x_1, c, (x_2, 0, d)) \end{aligned} \tag{A.6}$$

Figure A.1. Computation of $\mathrm{Ite}(f, g, h)$

Notes

1. The reader should not be surprised that a complex problem such as function-equivalence check can be solved in constant time once the RODGs for the functions are known. The process of building the RODGs involved may require, in itself, exponential time.
2. Strictly speaking, this result is only valid in spaces with an infinite number of concepts, because it is based on the dominance of the Solomonoff-Levin distribution over all semi-computable distributions. In this formalism, each concept can be defined by a string of symbols. The description length of a string (its Kolmogorov complexity) can be defined as the size of the smallest input to a three-tape Turing machine that causes it to write that string in the output tape. This result does not, therefore, contradict Schaffer (1994) or any other work that addresses the equivalence of all biases in spaces with a finite number of concepts. In practice, the description length as described above is not computable and one has to resort to less powerful languages to describe the concept. The underlying assumption is that, in many cases, the encoding scheme chosen is reasonably efficient and the computed complexity is a good approximation to the real value of the Kolmogorov complexity.
3. As pointed out by Quinlan and Rivest, the minimization of different linear combinations of d_g and d_d is also consistent with a Bayesian interpretation of the MDLP and may be chosen according to different beliefs about the concepts distribution. The algorithm can be set to minimize any linear combination of d_g and d_d, if that improves the performance in a particular set of problems. This choice of a different linear combination can be viewed as a way to compensate for inefficiencies in the encoding schemes chosen.
4. The first condition is only necessary to ensure the algorithm will terminate in a reasonable time. In most problems, a decision tree with a single node will always be obtained.

5. If there exists conflicting information in the training set, i.e., instances with the same values of the attributes but conflicting labels, these points are also considered as belonging to the *don't-care* set.
6. In the simplified description of Table 1, the procedure returns the first change that creates a decrease in the size of the RODG. As an alternative, the algorithm has the possibility of looking for the locally best incremental change. Very little difference in behavior was observed in the two modes and the mode described in table 1 is usually slightly faster.
7. Again, the procedure can be used in a slightly different form and return the best local change instead of the first one found. In this case, there is a significant performance penalty if a complete evaluation of the changes is performed instead of returning the first one with positive gain.
8. In this problem, the first 3 variables (the control variables) select which one of the following 8 (the data variables) defines the value of the output. Most algorithms for the generation of decision trees will test first the data variables, as they typically contain more information than the control variables. This results in a final decision tree that is much larger than the optimum solution.

References

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the 18th Annual ACM Symposium on the Theory of Computation* (pp. 273–282). Berkeley, CA: ACM Press.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, 24, 377–380.

Brace, K., Rudell, R., & Bryant, R. (1989). Efficient implementation of a BDD package. In *Proceedings of the Design Automation Conference* (pp. 40–45). Anaheim, CA: ACM Press.

Brayton, R. K., Hachtel, G. D., McMullen, C., & Sangiovanni-Vincentelli, A. S. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic Publishers.

Brayton, R. K., Hachtel, G. D., & Vincentelli, A. S. (1990). Multilevel logic synthesis. *Proceedings of the IEEE*, 78, 264–300.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35, 677–691.

Casella, G. & Berger, R. L. (1990). *Statistical Inference*. Pacific Grove, CA: Wadsworth & Brooks/Cole.

Coudert, O., Berthet, C., & Madre, J. C. (1989). Verification of synchronous sequential machines based on symbolic execution. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Volume 407 of *Lecture Notes in Computer Science* (pp. 365–373). Grenoble, France: Springer-Verlag.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 194–202). Tahoe City, CA: Morgan Kaufmann.

Fayyad, U. M. & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence* (pp. 1022–1027). Chambery, France: Morgan Kaufmann.

Friedman, S. J. & Supowit, K. J. (1990). Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39, 710–713.

Goldman, J. A. (1994). Machine learning: A comparative study of pattern theory and C4.5. Technical Report WL-TR-94-1102, Wright Laboratory, USAF, WL/AART, WPAFB, OH.

Ishiura, N., Sawada, H., & Yajima, S. (1991). Minimization of binary decision diagrams based on exchanges of variables. In *Proceedings of the International Conference on Computer Aided Design* (pp. 472–475). Santa Clara, CA: IEEE Computer Society Press.

Kam, T. & Brayton, R. (1990). Multi-valued decision diagrams. *UC Berkeley Tech. Report ERL M90/125*, EECS Department, Berkeley, CA.

Kohavi, R. (1994). Bottom-up induction of oblivious read-once decision graphs: Strengths and limitations. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 613–618). Tahoe City, CA: Morgan Kaufmann.

Li, M. & Vitányi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity*. New York, NY: Springer-Verlag.

Mahoney, J. J. & Mooney, R. J. (1991). Initializing ID5R with a domain theory: some negative results. Technical Report 91-154, CS Department, University of Texas at Austin, Austin, TX.

Meinel, C. (1989). *Modified Branching Programs and Their Computational Power.* New York, NY: Springer-Verlag.

Murphy, P. M. & Aha, D. W. (1991). *Repository of Machine Learning Databases - Machine readable data repository.* University of California, Irvine.

Oliveira, A. L. (1994). *Inductive Learning by Selection of Minimal Complexity Representations.* PhD thesis, UC Berkeley, Berkeley, CA. Also available as UCB/ERL Technical Report M94/97.

Oliveira, A. L. & Vincentelli, A. S. (1993). Learning complex Boolean functions : algorithms and applications. In *Advances in Neural Information Processing Systems 6* (pp. 911–918). Denver, CO: Morgan Kaufmann.

Oliveira, A. L. & Vincentelli, A. S. (1995). Inferring reduced ordered decision graphs of minimal description length. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 421–429). Tahoe City, CA: Morgan Kaufmann.

Oliver, J. J. (1993). Decision graphs - an extension of decision trees. Technical Report 92/173, Monash University, Clayton, Victoria, Australia.

Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5, 71–100.

Pearl, J. (1978). On the connection between the complexity and credibility of inferred models. *Journal of General Systems*, 4, 255–264.

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

Quinlan, J. R. (1993). *C4.5 - Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 227–248.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14, 465–471.

Rissanen, J. (1986). Stochastic complexity and modeling. *Annals of Statististics*, 14, 1080–1100.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In *Proceeddings of the International Conference on Computer Aided Design* (pp. 42–47). Santa Clara, CA: IEEE Computer Society Press.

Schaffer, C. (1994). A conservation law for generalization performance. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 259–265). New Brunswick, NJ: Morgan Kaufmann.

Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. *Transactions AIEE*, 57, 713–723.

Shiple, T. R., Hojati, R., Vincentelli, A. L. S., & Brayton, R. K. (1994). Heuristic minimization of BDDs using don't cares. In *Proceedings of the Design Automation Conference* (pp. 225–231). San Diego, CA: ACM Press.

Tani, S., Hamaguchi, K., & Yajima, S. (1993). The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proceedings of the Fourth International Symposium on Algorithms and Computation* (pp. 389–98). Hong Kong: Springer-Verlag.

Wallace, C. S. & Patrick, J. D. (1993). Coding decision trees. *Machine Learning*, 11, 7–22.

Yang, D. S., Rendell, L., & Blix, G. (1991). Fringe-like feature construction: A comparative study and a unifying scheme. In *Proceedings of the Eight International Conference on Machine Learning* (pp. 223–227). Evanston, IL: Morgan Kaufmann.

Received January 31, 1995
Accepted April 21, 1995
Final Manuscript March 6, 1996