Inferring Reduced Ordered Decision Graphs of Minimal Description Length

Arlindo L. Oliveira*
Alberto Sangiovanni-Vincentelli
Dept. of EECS, UC Berkeley, Berkeley CA 94720

July 1, 1994

Abstract

This work describes an approach for the inference of reduced ordered decision graphs from training sets. Reduced ordered decision graphs (RODGs) are graphs where the variables can only be tested in accordance with a pre-specified order and no redundant nodes exist. RODGs have several interesting properties that has made them the representation of choice for the manipulation of Boolean functions in the logic synthesis community.

We derive a RODG representation of the function implemented by a decision tree. This decision tree can be obtained from a training set using any one of the different algorithms proposed to date. This RODG is then used as the starting point for an algorithm that derives another RODG of minimal description length. The reduction in complexity is obtained by performing incremental changes in the RODG. By using ordered decision diagrams, the task of identifying common subgraphs is made much simpler than the identification of common sub-trees in a decision tree.

Ordered decision graphs require that a variable ordering be specified in advance. The algorithm that derives such an ordering is based on a reordering algorithm commonly used that finds a locally optimal ordering by swapping the order of two adjacent variables.

These algorithms are tested in a set of examples that are known to be hard to solve using decision trees. The results show that when an effective reduction of the description length is obtained, significant gains in generalization accuracy can be achieved. In all cases the generalization accuracy of the final RODG was better than the generalization accuracy of the decision tree that was used as the starting point.

1 Introduction

The induction of decision trees from labeled training set data [BFOS84, BFOS84, Qui86, QR89] has been a successful approach for the induction of classification rules. However, even though decision trees can, in principle, represent any concept, they have strong limitations since they are not concise representations for some concepts of interest. 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 high arity attributes are tested at a node. (See, for instance, [Oli93] for a more detailed description of these limitations)

Decision graphs (DG) have been proposed as one way to alleviate this problems, but the algorithms proposed to date for the construction of these graphs suffer from serious limitations. [MM91] proposed to identify related subtrees in a decision tree obtained using standard methods but reported limited success. This may have been caused by the fact that he used a non-canonical representation of Boolean functions (DNF expressions) which makes it hard to identify identical subtrees. [Oli93] proposed a greedy algorithm that performs either a join or a split operation, according to which one reduces more the description length. He reported some good results in relatively simple problems but our experiments using a similar approach failed in more complex test cases.

In this paper, we propose to use reduced ordered decision graphs (RODGs) as the underlying description. In RODGs, the variables can only be tested in accordance with a previously specified order. By using RODGs, we can

*207-194 Cory Hall, Dept. of EECS, UC Berkeley, Berkeley CA 94720. email: aml@ic.eecs.berkeley.edu

effectively avoid some of the problems encountered by other researchers when trying to identify common subtrees. We describe a simple but effective algorithm that derives an ordered decision graph with minimal description length from a decision tree built using standard techniques. The approach proposed here is very different from the one proposed in [Koh94] that also used RODGs. Although this approach performs well for small problems, it requires far too much computation to be applicable to any problems with a larger number of attributes.

Section 2 presents RODGs and briefly describes some of the interesting properties they exhibit. Section 3 describes the coding methods used and the cost function that is minimized. Section 4 describes how a RODG with a small description length can be obtained from a decision tree. Finally, some of the results obtained using this algorithm in problems extracted from the machine learning literature are described in section 5.

2 Decision Trees and Ordered Decision Graphs

We address the problem of inferring a classification rule given a set of objects, the training set. Each object is described in terms of a collection of discretely-valued attributes $(x_1 \in X_1, ..., x_n \in X_n)$ and the class it belongs to, $c \in C$. The objective is to infer a classification rule (a function $F : X_1 \times X_2 ... \times X_n \rightarrow C$) that can be used to classify unseen objects with the highest possible accuracy.

A decision graph is a rooted, directed, acyclic graph where each non-terminal node is labeled with the index of the attribute being tested at that node. Each terminal node is labeled with one label in C. From each non-terminal node an arc labeled with each possible value of the attribute being tested at that node leads to another node in the graph. Decision graphs can be used to classify any object by tracing a path from the root to a terminal node that follows the edges labeled with the values present in that object.

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 only one terminal node exists for each class, 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. The level of a node is defined as the position of the variable tested at that node in the variable ordering used.

Given an ordering, reduced ordered decision graphs are a canonical representation [Bry86], for functions in that domain. This means that given a function $F : X_1 \times X_2 ... \times X_n \rightarrow C$ and an ordering of the variables, there is one and only one representation for the function F.

Packages that manipulate reduced ordered decision graphs are widely available and have become the most commonly used tool for discrete function manipulation in the logic synthesis community [BHSV90]. Some of these packages are restricted to Boolean functions [BRB89] (each non-terminal graph has exactly two outgoing edges) while others [KB90] can accept multi-valued attributes.

3 Minimizing Message Length and Encoding of RODGs

Let t be the description length of a RODG and d be the length of the message required to describe the exceptions to this RODG in a given training set. According to the minimum description length principle (MDLP), one should choose the RODG that minimizes the total description length, $t + d$. However, as pointed out in [QR89], different linear combinations of t and d are also consistent with a Bayesian interpretation of the MDLP and may be chosen according to different beliefs about the data. Therefore, the algorithms described in section 4 aim at obtaining a decision graph that minimizes the cost function

$$f = \alpha t + \beta d \tag{1}$$

Setting $\alpha = \beta = 1$ one obtains the MDLP in its purest form.

Since we will be encoding RODGs using an approach inspired in the one proposed in [QR89] for decision trees, we briefly review their encoding scheme. In this reference, the encoding of a tree is defined recursively using the following rules:

1. A terminal node is encoded as a 0 followed by an encoding of the default class.
2. A non-terminal node is encoded starting with a 1, followed by the encoding of the variable tested at that node, followed by the encodings of the subtrees.

When encoding a RODG, we use the fact that some nodes will be visited more than once. Instead of re-describing them, we can simply make a reference to the description of that node.

1. A terminal node is encoded as a 0 followed by an encoding of the default class.
2. A non-terminal node that was never visited is encoded starting with 10 followed by an encoding of the variable tested at that node, followed by the encodings of the subgraphs.
3. A non-terminal node that was visited before is encoded starting with 11 followed by a reference to the (already described) function implemented by that node.

Following [QR89], 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). Also, when a reference to an already described function is given, only $\log_2(n)$ bits are needed, where n is the number of non-terminal functions described up to that point.

4 Obtaining a Small Decision Graph From a Decision Tree

For clarity and without loss of generality, we will henceforth assume that the attributes (or variable) are Boolean-valued and that there are only two classes. This considerably simplifies the description of the algorithm and doesn't represent a strong limitation, as any problem can be encoded using a simple binary code.

We assume for now that a variable ordering was defined. Section 4.3 details how this ordering is obtained.

4.1 Definitions

Let $B = \{0, 1\}$ and let the objects in the domain $A = B^n$ be described by a collection of Boolean attributes, $(x_1, \ldots, x_n)$. Let $T \subset A$ be the set of points in the input space that are in the training set. Let $|g|$, $(g : A \rightarrow B)$ denote the number of points in the input space for which function g is 1, and the operators $\wedge\vee$ and $\oplus$ stand for the Boolean *and*, *or* and *exclusive-or* of two Boolean functions, respectively.

Consider now a decision tree created using one of the approaches proposed in the literature (for example, [Qui86]). Let v_i denote the variable tested in node m_i of the decision tree, m_{t_i} (the *then* node) denote the node pointed to by the arc leaving node m_i when attribute v_i is 1 and m_{e_i} (the *else* node) denote the node pointed to by the arc leaving node m_i when attribute v_i is 0. Finally, let node m_0 be the root of the decision tree.

Each node in the decision tree implements a well-defined Boolean function, that has a unique representation in RODG form. Let $g_i : A \rightarrow B$ be the function implemented by tree node m_i. By definition, g_i can be obtained recursively using the Shanon cofactor operation, $g_i = \text{ITE}(v_i, g_{t_i}, g_{e_i}) = (\overline{v_i} \wedge g_{e_i}) \vee (v_i \wedge g_{t_i})$.

The RODG is built recursively by simply computing

$$g_0 = (\overline{v_0} \wedge g_{e_0}) \vee (v_0 \wedge g_{t_0}) \tag{2}$$

The recursion stops when both the *else* and *then* edges of the decision tree are terminal nodes.

Let the nodes of the RODG be $n_0, \ldots, n_k$ and let $f_i : A \rightarrow B$ be the function implemented by node n_i. As in the decision tree, n_{e_i} and n_{t_i} denote the nodes pointed by the *else* and *then* edges of node n_i. As before, v_i will identify the variable tested at node n_i. For each node n_i we will also define the function $w_i : A \rightarrow B$ that is 1 for the points in the training set that define paths in the RODG that go through node n_i.

The family of functions w_i can be computed recursively:

$$w_0(x) = \begin{cases} 1 & \text{if } x \in T \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$w_i = \Big(\bigvee_{n_j : n_{e_j} = n_i} \overline{v_j} \wedge w_j \Big) \vee \Big(\bigvee_{n_j : n_{t_j} = n_i} v_j \wedge w_j \Big) \tag{4}$$

Since the RODG obtained with expression 2 implements exactly the same function as the original decision tree, there is no gain in using it instead of the original tree to perform classification. However, the description length of the RODG obtained in this fashion can usually be reduced.

4.2 Reducing a RODG

The reduction in size of the RODG obtained from the decision tree is performed in steps. At each step, one or more nodes are removed from the RODG. The REMOVENODE procedure, described below, accepts as its argument one RODG and tries to reduce 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 tries to select, for each one of them, a different node n_k that implements a function as close to f_i as possible, for all the examples that reach n_i through that edge. If the RODG that results from redirecting each one of these edges has a cost function smaller than the original one, the procedure returns the modified RODG.

REMOVENODE(R)
  foreach n_i
    foreach n_j s.t. $n_{e_j} = n_i$ *For all nodes that have the else edge pointing to* n_i
      Select n_k such that $|(f_k \oplus f_i) \wedge w_j \wedge \overline{v_j}|$ is minimal
      Modify RODG such that $n_{e_j} = n_k$
    foreach n_j s.t. $n_{t_j} = n_i$ *For all nodes that have the then edge pointing to* n_i
      Select n_k such that $|(f_k \oplus f_i) \wedge w_j \wedge v_j|$ is minimal
      Modify RODG such that $n_{e_j} = n_k$
    if Modified RODG has smaller cost function
      return (Modified RODG)
  return (Failure)

If the above procedure fails to make one node redundant, procedure REPLACEPAIR is called. This procedure also takes as its argument one RODG. REPLACEPAIR tries to remove a pair of nodes by creating another node that implements a function as close as possible to the functions implemented by the pair of nodes under consideration. However, the value of this function is only relevant for the examples that reach these nodes.

REPLACEPAIR(R)
  foreach n_i
    foreach n_j
      $s := (w_i \wedge f_i) \vee (w_j \wedge f_j)$ *s contains the desired state of the new node*
      $w := w_i \vee w_j$ *w is 1 for all objects that reach nodes* n_i *or* n_j
      Create n_k =ITE(v_k, n_a, n_b) such that $|(f_k \oplus s) \wedge w|$ is minimal
      Modify RODG such that incoming edges into n_i and n_j point to n_k
      if Modified RODG has smaller cost function
        return (Modified RODG)
  return (Failure)

The main loop simply calls the above procedures until both return failure, while calling, at each iteration, the reordering procedure described in section 4.3.

MAINLOOP()
  S := RODGFROMTREE()
  repeat
    R := S
    R := REORDER(R)
    S := REMOVENODE(R)
    if S = Failure
      S := REPLACEPAIR(R)
  until S = Failure
  return (R)

For large problems, the procedure REPLACEPAIR can be time consuming and the user has the option of not using it. A more efficient version of the algorithm can be obtained in the degenerate case when $\beta >> \alpha$ in (1). In this case, it is possible to test for the improvement in the cost function in a specially efficient way because the above condition forces the resulting RODG to perfectly classify all the data in the training set.

4.3 Selecting the Best Ordering

The selection of a good ordering for the variables is of critical importance if the target is to obtain a compact RODG. Regrettably, selecting the optimal ordering for a given function is NP-hard and cannot be solved exactly in many cases. However, many heuristic algorithms have been proposed for this problem [FS90, Rud93].

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 original RODG obtained from the decision tree.

Our implementation uses the *sift* algorithm for dynamic RODG ordering [Rud93]. A complete description of this algorithm is impossible here due to space limitations. However, the basic ideas underlying the algorithm are straightforward. It is known that, given a RODG, swapping the order of two adjacent variables can be done very efficiently because only the nodes in those 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 change in the RODG is performed.

5 Experiments

To evaluate the efficacy of the approach, we selected a set of problems that are known to be hard to learn using a decision tree representation. All but one of problems used were proposed in [TBB+91], [PH90] and [MA91]. *Kkp* is an encoding of king-pawn versus king chess endings. A brief description of all the problems in given in appendix A. For all problems, multi-valued attributes were encoded using a binary encoding. For each problem, we performed 10 runs with randomly generated training sets. Since it is known that the training sets are error free, we ran the algorithm under the condition $\beta >> \alpha$ (see section 4), as the current version of the algorithm can only handle the general case for problems smaller than the ones used here. This implies that, for all problems, the resulting RODG classified correctly 100% of the examples in the training set.

The resulting decision tree and RODG were tested using one independent test set (that included, possibly, some or all of the examples present in the training sets). Statistics about the problems and the sizes of the training and test sets used are listed in table 1.

Problem	# bin. inputs	training	testing
Monks2	10	216	432
Dnf2	40	2187	2000
Dnf3	32	1652	2000
Par4	16	640	2000
Mux11	32	802	2000
Kkp	17	250	2000
TicTacToe	18	479	958

Table 1: Problem statistics

The graphs in figure 1 show the evolution of the classification accuracy as the description length of the RODG is reduced. For the problems *monks2*, *mux11*, *dnf3* and *kkp*, we plotted the accuracy in the test set versus the description length of the successives RODGs obtained by the algorithm described in section 4. For each line, the point furthest to the right is obtained with the RODG derived directly from the decision tree and the accuracy at that point is the same as the one obtained with the decision tree. The leftmost points of each line correspond to the last RODG obtained. The figure shows that, for these problems, a significant increase in classification accuracy is always obtained by reducing the size of the RODG.

Table 2 lists the results obtained for all the problems. The second and third columns list the description length of the original decision tree and the first RODG. The fourth column lists the description length of the final RODG obtained. These values are computed as described in section 3. The following two columns list the generalization

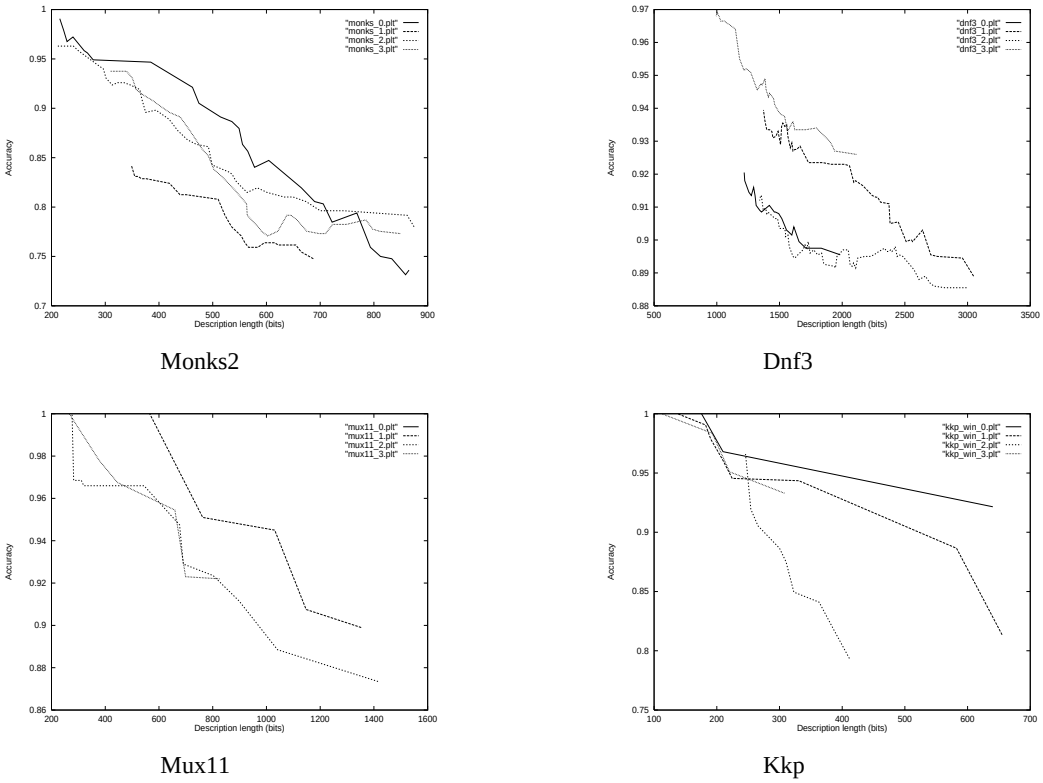


Figure 1: Accuracy versus description length

accuracy of the first and last RODG. Finally, the last column lists the improvement in generalization accuracy obtained for each example, together with the standard deviations.

Problem	Description length (bits)			Generalization accuracy		
	Dec Tree	Init RODG	Final RODG	DT	RODG	Improv $\pm\sigma$
Monks2	567.3	829.6	232.9	75.48	95.81	20.33 ± 4.34
Dnf2	1552.3	3777.8	2103.1	89.13	90.77	1.64 ± 0.78
Dnf3	1088.2	2318.0	1269.7	89.89	92.54	2.65 ± 1.65
Par4	1580.0	4981.8	64.64	59.09	100.0	40.91 ± 4.19
Mux11	788.6	968.20	135.40	90.92	100.0	9.08 ± 4.82
Kkp	393.5	490.3	110.5	81.24	98.25	17.01 ± 3.29
TicTacToe	340.7	483.2	364.7	96.18	96.70	0.51 ± 0.71

Table 2: Description lengths and accuracy

There are several interesting points that should be noted. First, the description of the RODG that results from encoding the decision tree usually requires more bits than the original decision tree. This is more a consequence of the ordering requirement than the slightly less efficient encoding of each node when no reconvergence exists in the graph.

The reduction performed by the algorithm can vary widely, from almost no gain to several orders of magnitude. Not surprisingly, the larger gains in classification accuracy are obtained when more compression is achieved in the description of the RODG.

In all examples the accuracy of the final RODG was, on average, better than the accuracy of the decision tree. However, the improvement observed for the TicTacToe problem is not statistically significant.

Finally, figures 2 and 3 show the first and last RODGs obtained in the first run of the *mux11* and the *kkp* problems.

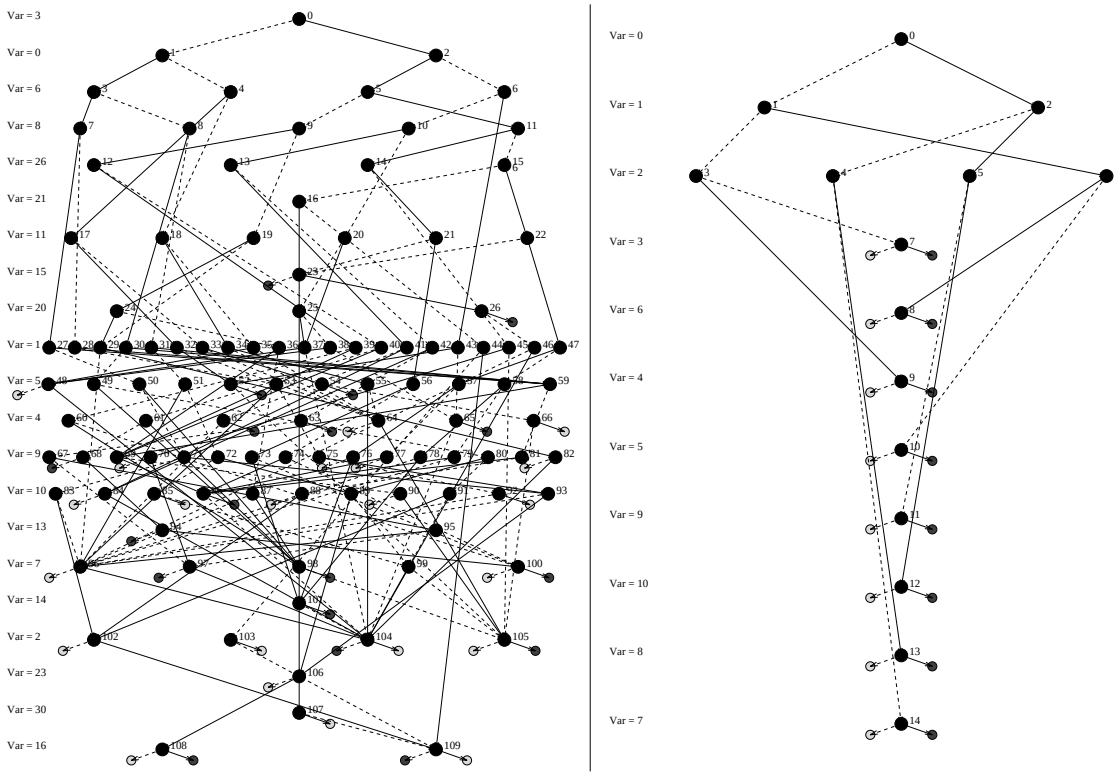


Figure 2: First and last RODG for the first *mux11* run

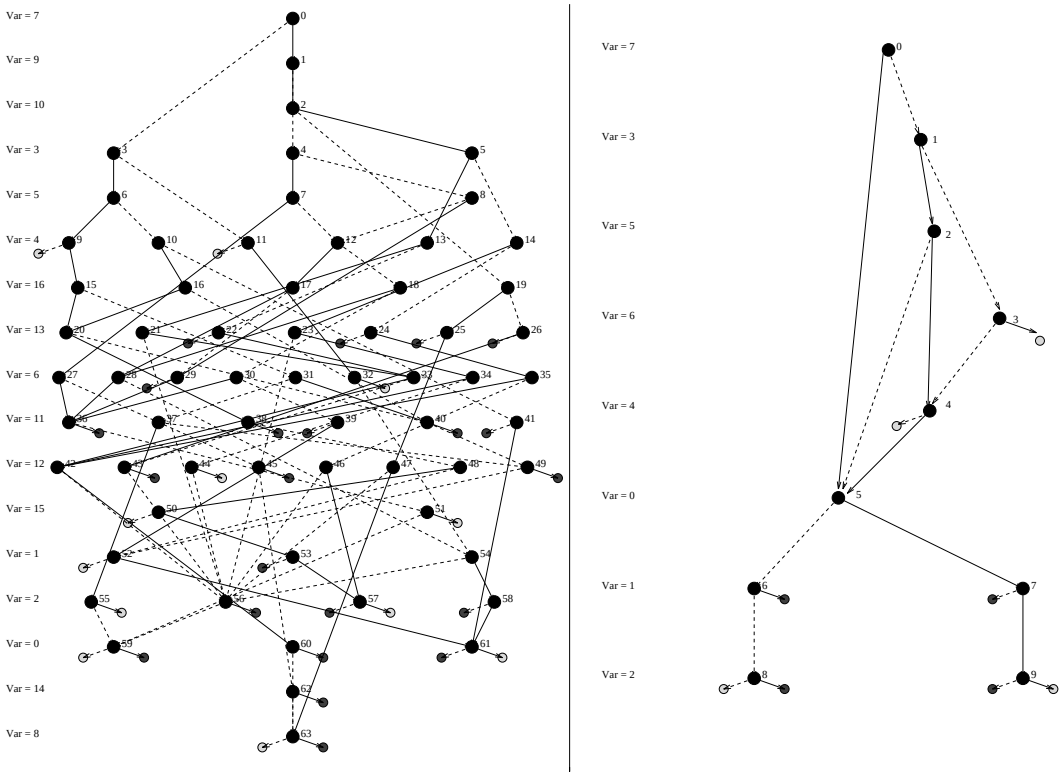


Figure 3: First and last RODG for the first *kkp* run

The interesting point to note, apart from the reduction in size, is the fact that the ordering algorithm also obtained an optimal order of the variables. Decision trees have a hard time in problems like *mux11* (a 3-bit multiplexer with 21 irrelevant variables) because, as pointed out in [Qui88], the greedy strategy for node splitting tends to favor testing the data attributes first instead of the selection bits.

6 Conclusions and Future Work

The results presented in section 5 show that the accuracy of the generalization performed by a decision tree algorithm can be improved, in some cases significantly, by using the algorithms described in section 4.

Current research is concentrated on the improvement of the algorithm, both in the quality of the results obtained and the execution time requirements.

Finally, it must be stated that many other good alternatives have been proposed that aim at increasing the quality of the generalization performed by a decision tree. Both constructive induction techniques and different forms of post-processing of the decision tree have been proposed. Specially worth mentioning are the *fringe*-like algorithms [PH90, YRB91, OSV93] that incrementally build new composite attributes by looking at the tests performed at the fringes of decision trees. Our own experiments with these algorithms have shown that they are remarkably efficient at learning concepts that accept a compact DNF representation. We conjecture that the performance of the approach described in this paper cannot beat this type of algorithms in these problems. Furthermore, it is know that there are functions that have compact DNF representations but require exponentially large RODGs, no matter what ordering is selected [Dev93]. *Fringe*-like algorithms, however, have a much harder time when the problems at hand do not have a compact DNF representation. In that case, we believe an approach like the one outlined here can be one of the best alternatives available. Ultimately, the MDLP should be used to evaluate which of the representations is more adequate for the problem at hand.

It must be noted, however, that using RODGs instead of unordered decision graphs comes at a price, since there are functions that cannot be represented in compact form in RODG form but accept compact representations if the fixed variable ordering requirement is not enforced. We believe this limitation if a small price to pay for the superior ease of manipulation that is obtained when one uses RODGs.

A Description of the Problems Used

The problems *Dnf2*, *Dnf3*, *Par4* and *Mux11* were proposed in [PH90] and can be described by the following Boolean formulas:

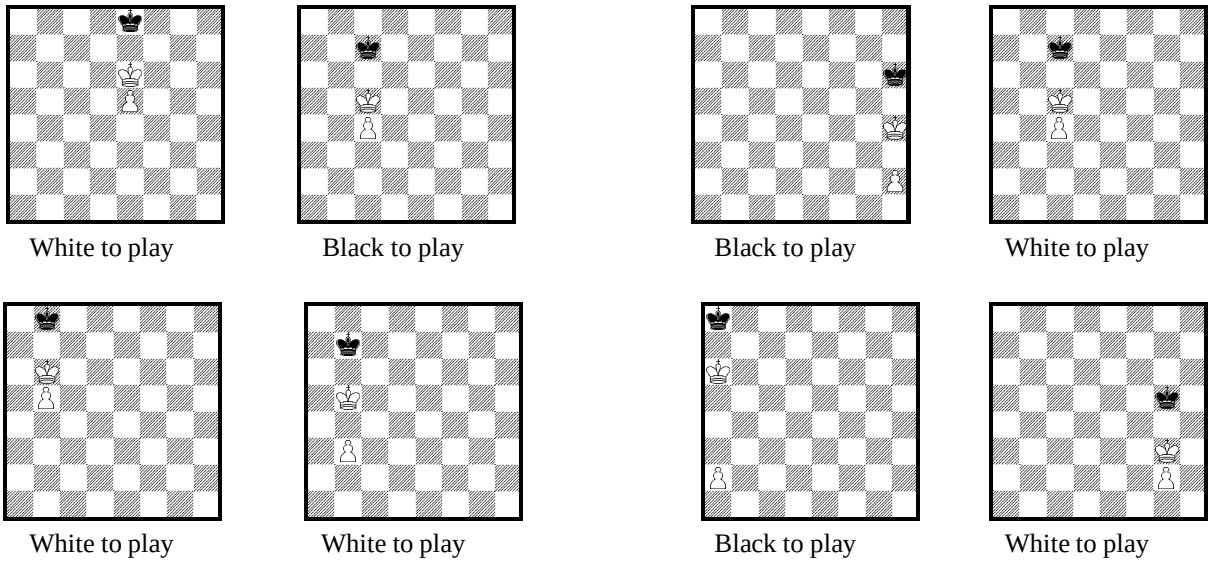
- dnf2 : $f(x_1 \ldots x_{32}) = x_1x_3x_{14}x_{19}x_{26}x_{35}x_{36} \vee x_8x_{15}x_{31}x_{37} \vee x_5x_{10}x_{14}x_{27}x_{29} \vee x_{18}x_{20}x_{30}x_{36} \vee x_2x_3x_9x_{19}x_{24} \vee x_{24}x_{25}x_{27}x_{36}x_{37} \vee x_6x_7x_{14}x_{25}x_{26}x_{31}x_{34} \vee x_1x_6x_{22}x_{30}$
- dnf3 : $f(x_1 \ldots x_{32}) = x_1x_2x_6x_8x_{25}x_{28}\overline{x_{29}} \vee x_2x_9x_{14}\overline{x_{16}}\,\overline{x_{22}}x_{25} \vee x_1\overline{x_4x_{19}x_{22}}x_{27}x_{28} \vee \overline{x_2}x_{10}x_{14}\overline{x_{21}x_{24}} \vee x_{11}x_{17}x_{19}x_{21}\overline{x_{25}} \vee \overline{x_1x_4}x_{13}\overline{x_{25}}$
- par4 : $f(x_1 \ldots x_{16}) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$
- mux11 : $f(x_1 \ldots x_{32}) = \overline{x_1x_2x_3}x_4 \vee \overline{x_1x_2}x_3x_5 \vee \overline{x_1}x_2\overline{x_3}x_6 \vee \overline{x_1}x_2x_3x_7 \vee x_1\overline{x_2x_3}x_8 \vee x_1\overline{x_2}x_3x_9 \vee x_1x_2\overline{x_3}x_{10} \vee x_1x_2x_3x_{11}$

Problem *Monks2* was proposed in [TBB+91] and is the encoding of a concept in a hypothetic robot world. *TicTacToe* results from the encoding of TicTacToe endings using 3 distinct symbols for each square. The target class defines whether the result is a draw or a win.

The problem *Kkp* results from the encoding of a *King+Pawn* versus *King* chess endings. This problem differs from similar problems proposed previously in the machine learning literature [Qui83] in the fact that no higher level attributes were. Instead, the board coordinates of the chess pieces were used directly.

The positions under study are the white *King+Pawn* versus black *King* chess endings when both Kings are in front of the white pawn. The coordinates were encoded using a binary code and an extra binary variable describes who is to play. The problem was made harder by adding 8 extra binary variables that are random.

The figure below shows some examples of the positions used. The ones on the left represent a forced win for white, while the ones on the right are a draw.



White to play | Black to play | Black to play | White to play

White to play | White to play | Black to play | White to play

References

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[BHSV90] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, 78:264–300, 1990.

[BRB89] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conference*, June 1989.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 1986.

[Dev93] S. Devadas. Comparing two-level and ordered binary decision diagrams representations of logic functions. *IEEE Transaction on Computer-Aided Design*, 12(5), 1993.

[FS90] Steven J. Friedman and Kenneth J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Trans. Comput.*, 39(5):710–713, May 1990.

[KB90] T. Kam and R.K. Brayton. Multi-valued decision diagrams. *Tech. Report No. UCB/ERL M90/125*, December 1990.

[Koh94] Ron Kohavi. Bottom-up induction of oblivious read-once decision graphs. In *European Conference in Machine Learning*, 1994.

[MA91] P. M. Murphy and D. W. Aha. *Repository of Machine Learning Databases - Machine readable data repository*. University of California, Irvine, 1991.

[MM91] J. J. Mahoney and R. J. Mooney. Initializing ID5R with a domain theory: some negative results. Technical Report 91-154, CS Dept., University of Texas at Austin, Austin, TX, 1991.

[Oli93] J. J. Oliver. Decision graphs - an extension of decision trees. Technical Report 92/173, Monash University, Clayton, Victoria 3168, Australia, 1993.

[OSV93] Arlindo L. Oliveira and A. Sangiovanni-Vincentelli. Learning complex boolean functions : Algorithms and applications. In *Advances in Neural Information Processing Systems 6*, Denver, CO, 1993. Morgan Kaufmann.

[PH90] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–100, 1990.

[QR89] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the Minimum Description Length Principle. *Inform. Comput.*, 80(3):227–248, March 1989. (An early version appeared as MIT LCS Technical report MIT/LCS/TM-339 (September 1987).).

[Qui83] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J.G. Carbonelli, and T. M. Mitchell, editors, *Machine Learning – An Artificial Intelligenc Approach*, pages 463–482. Springer-Verlag, 1983.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui88] J. R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Fifth International Conference on Machine Learning*, pages 135–141, 1988.

[Rud93] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD*, pages 42–47. IEEE Computer Society Press, 1993.

[TBB+91] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. de Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufaman, S. Keller, I. Kononenko, J. Kreusiger, R. S. Michalski, T. Mitchell, P. Pachowitz, Y. Reich, H. Vafaic, W. Van de Weldel, W. WEnzel, J. Wnek, and J. Zhang. The monk's problems: a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[YRB91] D. S. Yang, L. Rendell, and G. Blix. Fringe-like feature construction: A comparative study and a unifying scheme. In *Proceedings of the Eight International Conference in Machine Learning*, pages 223–227, San Mateo, 1991. Morgan Kaufmann.