Appears in AAAI-94

# Bottom-Up Induction of Oblivious Read-Once Decision Graphs: Strengths and Limitations

**Ron Kohavi**
Computer Science Department
Stanford University
Stanford, CA 94305
ronnyk@CS.Stanford.EDU

**Abstract**

We report improvements to HOODG, a supervised learning algorithm that induces concepts from labelled instances using oblivious, read-once decision graphs as the underlying hypothesis representation structure. While it is shown that the greedy approach to variable ordering is locally optimal, we also show an inherent limitation of all bottom-up induction algorithms, including HOODG, that construct such decision graphs bottom-up by minimizing the width of levels in the resulting graph. We report our empirical experiments that demonstrate the algorithm's generalization power.

## Introduction

In supervised classification learning, one tries to find a structure, such as a decision-tree, a neural net, or a Boolean formula, that can be used to accurately predict the label of novel instances. A given concept can be represented by different structures that differ in many aspects, including comprehensibility, storage size, and query time.

Decision trees provide one structure that is commonly constructed using top-down induction techniques (Quinlan 1993; 1986; Moret 1982). However, the tree structure used to represent the hypothesized target concept suffers from some well-known problems, most notably the replication problem and the fragmentation problem (Pagallo & Haussler 1990). The replication problem forces duplication of subtrees in disjunctive concepts such as $(A \wedge B) \vee (C \wedge D)$; the fragmentation problem causes partitioning of the data into fragments, when a high-arity attribute is tested at a node. Both problems reduce the number of instances at lower nodes in the tree—instances needed for statistical significance of tests performed during the tree construction process.

In (Kohavi 1994), Oblivious read-Once Decision Graphs (OODGs) were introduced as an alternative representation structure for supervised classification learning. OODGs retain most of the advantages of decision trees, while overcoming the two problems mentioned above. OODGs are similar to Ordered Binary Decision Diagrams (OBDDs) (Bryant 1986), which have been used in the engineering community to represent state-graph models of systems, allowing verification of finite-state systems with up to $10^{120}$ states (Burch, Clarke, & Long 1991). We refer the reader to (Kohavi 1994) for a discussion of related work.

OODGs have a different bias from that of decision trees, and thus some concepts that are hard to represent as trees are easy to represent as OODGs, and vice-versa. Since OODGs are graphs, they are easy for humans to perceive, and should be preferred over other representations (*e.g.*, neural nets) whenever it is important to comprehend the meaning and structure of the induced concept.

In this paper, we investigate the strength and limitations of inducing OODGs bottom-up using HOODG, a greedy hill-climbing algorithm for inducing OODGs, previously introduced in (Kohavi 1994). We show that on the one hand, a greedy choice of variables always yields an ordering that is locally optimal for fully specified functions and cannot be improved by a single exchange of adjacent variables (a technique sometimes used in the engineering community). On the other hand, there is an optimization step in the algorithm that is shown to be intractable, unless P=NP.

In the next two sections, we describe the OODG structure, some of its properties, and the framework of the bottom-up induction algorithm. We then describe the HOODG algorithm, discuss the heuristics and improvements made since its introduction, and new theoretical results. We follow with the experimental results, and conclude with a summary and discussion of future work.

## Oblivious Read-Once Decision Graphs

In this section, we formally define the structure of decision graphs and then specialize it to oblivious, read-once decision graphs (OODGs).

Given $n$ discrete variables (attributes), $X_1$, $X_2$, ..., $X_n$, with domains $D_1$, ..., $D_n$ respectively, the **instance space** $\mathcal{X}$ is the cross-product of the domains, *i.e.*, $D_1 \times \cdots \times D_n$. A ***k*-categorization function** is a function $f$ mapping each instance in the instance space to one of $k$ categories, *i.e.*, $f : \mathcal{X} \mapsto \{0, \ldots, k-1\}$. Without loss of generality, we assume that for each category there is at least one instance in $\mathcal{X}$ that maps to it.

A **decision graph** for a $k$-categorization function over variables $X_1, X_2, \ldots, X_n$ with domains $D_1, D_2, \ldots, D_n$, is a directed acyclic graph (DAG) with the following properties:

1. There are exactly $k$ nodes, called **category nodes**, that are labelled $0, 1, \ldots, k-1$, and have outdegree zero.
2. Non-category nodes are called **branching nodes.** Each branching node is labelled by some variable $X_i$ and has $|D_i|$ outgoing edges, each labelled by a distinct value from $D_i$.
3. There is one distinguished node—the **root**—that is the only node with indegree zero.

The category assigned by a decision graph to a given variable assignment (an instance), is determined by tracing the unique path from the root to a category node, branching according to the labels on the edges.

In a **read-once** decision graph, each variable occurs at most once along any computation path. In a **levelled** decision graph, the nodes are partitioned into a sequence of pairwise disjoint sets, the levels, such that outgoing edges from each level terminate at the next level. An **oblivious** decision graph is a levelled graph such that all nodes at a given level are labelled by the same variable. An oblivious decision graph is **reduced** if there do not exist two distinct nodes at the same level that branch in exactly the same way on the same values. If two such nodes exist, they can be united.

An **OODG** is a reduced oblivious, read-once decision graph. The **size** of an OODG is the number of nodes in the graph, and the **width** of a level is the number of nodes at that level. A **constant node** is a node, such that all edges emanating from it, terminate at the same node of the subsequent level. Figure 1 shows an OODG for 3-bit parity with one totally irrelevant attribute.

OODGs have many interesting properties. These include: an OODG is canonical for any total function if an ordering on the variables is given; any symmetric function (*e.g.*, parity, majority, and $m$ of $n$) can be represented by an OODG of size $O(n^2)$; and the width of levels in OODGs is bounded by $\min\left\{2^i, k^{2^{(n-i)}}\right\}$ for Boolean inputs. We refer the reader to (Kohavi 1994) for a more detailed description of these properties.

## Bottom-Up Construction of OODGs

In this section we present an algorithm for constructing a reduced OODG given the full (labelled) instance space. The algorithm is recursive and nondeterministic. For simplicity of notation, we assume Boolean variables and an arbitrary number of categories.

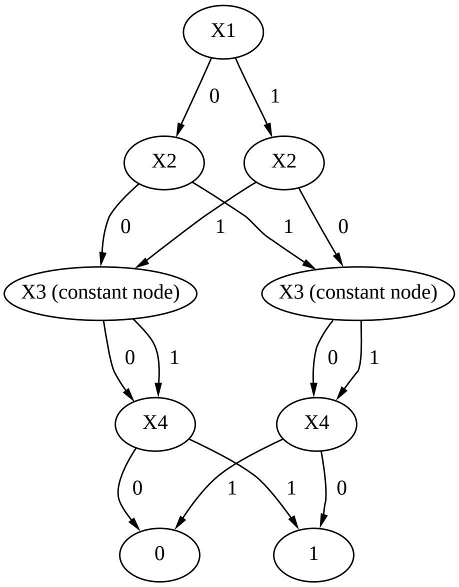


Figure 1: An OODG for 3-bit parity:
$f = X_1 \oplus X_2 \oplus X_4$ .
($\oplus$ denotes exclusive-or, $X_3$ is totally irrelevant.)

The input to the algorithm is a set of sets, $\{C_0, C_1, \ldots, C_{k-1}\}$, where each set $C_i$ is the set of all instances labelled with category $i$. The output of the algorithm is an OODG that correctly categorizes the training set.

The algorithm, shown in Figure 2, creates sets of instances, such that each set corresponds to one node in the graph (the input sets corresponding to the category nodes). Intuitively, we would like an instance in a set $C_i$ to reach node $V_i$ (corresponding to the set), when the instance's path is traced from the root of the completed OODG, branching at branching nodes according to the attribute values.

Given the input, the algorithm nondeterministically selects a variable $X$ to test at the penultimate level of the OODG. It then creates new sets of instances (corresponding to the nodes in the penultimate level of the final OODG), which are projections of the original instances with variable $X$ deleted. The sets are created so that a set $C'_{xy}$ (which matches a branching node $V'_{xy}$) contains all projections of instances that are in $C_x$ when augmented with $X = 0$, and in $C_y$ when augmented with $X = 1$. In the graph, the branching node corresponding to $C'_{xy}$ will have the edge labelled 0 terminating at node $V_x$, and the edge labelled 1 terminating at node $V_y$.

The new sets now form a smaller problem over $n-1$ variables, and the algorithm calls itself recursively to compute the rest of the OODG with the nonempty sets of the new level serving as the input. The recursion stops when the input to the algorithm is a single set, possibly consisting of the *null* instance (0 variables).

Input: $k$ sets $C_0, \ldots, C_{k-1}$, such that $\mathcal{X} = \bigcup_{i=0}^{k-1} C_i$ (the whole instance space).
Output: Reduced OODG correctly categorizing all instances in $\mathcal{X}$.

1. If $(k = 1)$, then return a graph with one node.
2. Nondeterministically select a variable $X$ to be deleted from the instances.
3. Project the instances in $C_0, \ldots, C_{k-1}$ onto the instance space $\mathcal{X}'$, such that variable $X$ is deleted. Formally, if $X$ is the $i$th variable,
$$\mathcal{X}' \leftarrow \pi_{(X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n)} \bigcup_{i=0}^{k-1} C_i \quad (\text{where } \pi_{(\vec{x})} \text{ means project on } \vec{x}) \; .$$
4. For all $i, j \in \{0, \ldots, k-1\}$, let $C'_{ij}$ be the set containing instances from $\mathcal{X}'$ such that the instances are in set $C_i$ when augmented with $X = 0$, and in $C_j$ when augmented with $X = 1$. Formally,
$$C'_{ij} = \left\{ \langle X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n \rangle \;\middle|\; \begin{array}{l} \langle X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n \rangle \in C_i \text{ and} \\ \langle X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n \rangle \in C_j \end{array} \right\} \; .$$
5. Let $k'$ be the number of non-empty sets from $\{C'_{ij}\}$. Call the algorithm recursively with the $k'$ non-empty sets, and let $G$ be the OODG returned.
6. Label the $k'$ leaf nodes of $G$, corresponding to the non-empty sets $C'_{ij}$ with variable $X$. Create a new level with $k$ nodes corresponding to the sets $C_0, \ldots, C_{k-1}$. From the node corresponding to each $C'_{ij}$, create two edges: one labelled 0, terminating at the category node corresponding to $C_i$, and the other labelled 1, terminating at the category node corresponding to $C_j$.
7. Return the augmented OODG.

Figure 2: A nondeterministic algorithm for learning OODGs.

## HOODG: A Hill Climbing Algorithm For Constructing OODGs

In this section, we address the two problems ignored in the algorithm previously described:

1. Ordering the variables for selection (Step 2).
2. If the full instance space is not available, the set $C'_{ij}$ in Step 4 may be not uniquely defined, and there may be many sets consistent with the projection.

### Ordering the Variables

Given the full instance space, it is possible to find the optimal ordering using dynamic programming, by checking $2^n$ different orderings (Friedman & Suppowit 1990). Since this is impractical in practice, our implementation greedily select the variable that yields the smallest width at the next level, excluding constant nodes. We break ties in favor of minimizing the number of edges. If all nodes are constant nodes (the attribute is deemed irrelevant), we do another lookahead step and pick the variable that maximizes the number of irrelevant attributes at the next level.

This heuristic is different from the original one proposed in (Kohavi 1994). The main difference is the fact that the minimization is done excluding constant nodes. By ignoring constant nodes, the algorithm scales better when the target concept is decomposable. For example, suppose the target concept can be decomposed into a disjunction of three subproblems on disjoint variables, such as the following target concept:
$$f = (A \wedge B \wedge C) \vee (D \wedge E) \vee (F \wedge G)$$
An OODG with leaves **0** and **1** that implements $A \wedge B \wedge C$ can be extended by connecting an OODG that implements the other two subproblems into the **0** leaf. The **1** leaf can be made into a series of constant nodes at the lower levels until the **1** node of the final OODG is reached. When the bottom-up construction takes place, we would like to take into account only the number of non-constant nodes, as these indicate the actual dependencies on the variable.

The following theorem shows that in a bottom-up construction, where the full instance space is available, the above heuristic is locally optimal and cannot be improved by a single exchange of neighboring variables. Such exchanges were done to improve the size of OBDDs when created top-down in (Fujita, Matsunaga, & Kakuda 1991).

**Theorem 1** *If during a bottom-up construction, the variable that creates the smallest width at each level is chosen, no exchange of two adjacent variables will improve the size of the OODG or OBDD.*

The proof is based on the fact that exchanging neighboring variables changes the number of nodes at only one of the two levels.

### Incomplete Projections

If the full instance space is not given, there will be projections of instances for which some values of the deleted attribute will be missing (*e.g.*, a projected instance must branch to some node on values 0 and 2, but the destinations for values 1 and 3 are unknown).

Call such projections *Incomplete Projections,* or **IPs.** Assigning values to the missing destinations of the IPs constitutes a bias, since it determines how unseen instances will be classified.

Following Occam's razor, we would like to find the smallest OODG consistent with the data (we assume no noise). We are thus looking for a minimal set of branching nodes that "covers" all projections, *i.e.*, a minimal cover.

An IP is **consistent** with another projection, P (at the same level of the graph), if they do not have conflicting destinations on the same value of the deleted variable. An IP is **included** in another projection, P, if they are consistent, and if all destinations defined for the IP are also defined for the projection P. (Note that *included* is an asymmetric relation.)

The greedy strategy for assigning incomplete projections to nodes, starts creating projection sets (branching nodes) from projections having the greatest number of known destinations, and then from projections with fewer known destinations. Following a least commitment strategy, each projection is placed in a projection set it is included in, whenever possible (hence not forcing a new destination); otherwise, it is placed in a set where it is consistent with all instances, if possible; otherwise, a new projection set is created, consisting of the single projection.

Our heuristic breaks ties in favor of projection set that has the most instances differing by at most one bit, and given equality, breaks ties in favor of adding the minimum number of new destinations (again, least commitment). These tie-breakers were added to the original heuristic, presented in (Kohavi 1994), after it was noted that there are many cases where they are needed. We have tried different tie-breaking heuristics, and many reasonable heuristics perform better than the arbitrary tie-breaking originally used.

As the following results show, it is unlikely that an algorithm finding the smallest consistent OODG will be found, even for a given ordering. In (Takenaga & Yajima 1993), it was shown that identifying whether there exists an OBDD with $k$ nodes that is consistent with labelled instances is NP-complete, and this result applies to OODGs too. The following theorem shows that minimizing even a single level in an OBDD or OODG is NP-complete:

**Theorem 2 (Hardness of minimal projection)** *The following decision problem is NP-complete:*
*Given a set of labelled instances, an ordering on the variables, and two positive integers $w$ and $\ell$; is there an OODG (or OBDD) that has width $\leq w$ at level $\ell$, and that correctly classifies all instances?*

The reduction was done from graph $k$-colorability (chromatic number) using only Boolean variables. This is a strong negative result, since it is known that the chromatic number of a graph cannot be approximated to within any constant multiplicative factor unless P=NP (Lund & Yannakakis 1993). Note that this result applies to any algorithm that attempts to minimize the width of an OODG at a given level, whether done incrementally as in HOODG, or otherwise.

## Experimental Results

We now turn to a series of experiments that attempt to evaluate the performance of the HOODG algorithm. Table 1 shows the accuracy results[1] for ID3, C4.5 (Quinlan 1993), Oliver's decision graph algorithm, DGRAPH (Oliver 1993), and HOODG, on the following datasets that we generated or retrieved from (?):

**Monk 1,2** In (Thrun *et al.* 1991), 24 authors compared 25 machine learning algorithms on problems called the monks problems. In this domain there are six attributes with discrete values. The Monk 1 problem has a single training set, but it is too easy. To make the problem harder, we ran the algorithms on 10 sets of 60 instances each—about half of the original training set. The test set is the whole space.

In the Monk 2 problem, we ran the algorithms on 10 sets of 169 instances each, the same size as the original training set. The problem is very hard, but becomes easier under local encoding, where each attribute value is assigned an indicator variable. This encoding was used for neural networks in the comparison.

**Parity** The target concept is the parity of 5 bits out of 10 bits with 5 (uniformly random) irrelevant bits. We averaged 10 sets of 100 instances each.

**Vote** The vote database includes votes for each of the U.S. House of Representatives Congressmen on 16 key votes. The classes are Democrat and Republican. There are 435 instances with duplicates. We ran ten-fold cross-validation.

**Breast** In the Wisconsin breast-cancer database, the task is to predict reoccurrence or non-reoccurrence of breast-cancer sometime after an operation. There are nine attributes, each with 10 discretized values. There are 699 instances with duplicates. We ran tenfold cross validation on the original encoding and then using binary encoding.

The worst-case time complexity of the HOODG algorithm is $O(ns^2 + is^2(n-1))$ per level, where $i$ is the number of irrelevant attributes at the given level, and $s$ is the number of projected instances at that level. This assumes that the number of values per attribute

[1] C4.5 runs were made with `-m 1`, and the better result of running with and without the `-s` flag for grouping. Oliver's DGRAPH was run with 2 levels of lookahead and with different $p$ values (we give the one that yields the minimum message length as suggested by Oliver). HOODG was run without the 2-level lookahead on irrelevant attributes for the Breast-cancer database because the lookahead was too expensive time-wise.

| Data Set | ID3 | C4.5 | DGRAPH | HOODG |
|---|---|---|---|---|
| Monk 1 (60/432) | 80.32% ± 6.45% | 83.56% ± 9.27% | 73.89% ± 2.68% | ✓★ 100.00% ± 0.00% |
| Monk 2 (169/432) | 69.17% ± 2.01% | 72.73% ± 5.66% | 66.67% ± 1.46% | ✓★ 91.30% ± 1.98% |
| Monk 2 local | 78.08% ± 6.66% | 75.75% ± 7.83% | 67.13% ± 0.00% | ✓★ 99.14% ± 0.62% |
| Parity (100/1024) | 54.00% ± 2.68% | 51.56% ± 2.32% | 50.00% ± 0.00% | ✓★ 100.00% ± 0.00% |
| Vote (435/XV) | 93.57% ± 4.00% | ✓ 95.43% ± 4.31% | ✓★ 95.63% ± 3.69% | ✓ 94.03% ± 3.46% |
| Breast (699/XV) | ✓ 94.42% ± 4.68% | ✓★ 94.64% ± 6.16% | ✓ 93.85% ± 4.26% | 86.99% ± 6.50% |
| Breast binary | ✓ 94.01% ± 4.57% | ✓★ 96.07% ± 6.16% | 92.84% ± 5.94% | ✓ 95.03% ± 2.77% |

Table 1: Comparison of different algorithms. Results are averages of 10 runs with standard deviation after the ± sign. Number in parentheses denote the training set size and test set size; XV means ten-fold cross validation. "local" means local encoding, "binary" means binary encoding. The best accuracy for each dataset is shown with a star (★), and accuracies within one half standard deviation of the best, are marked with a checkmark (✓). Such small differences in accuracy indicate comparable performance.

is a bounded constant. If we ignore the two-level lookahead for irrelevant attributes, the time complexity of the overall algorithm is $O(n^2m^2)$, where $m$ is the number of instances in the training set.

Running on a SPARCstation ELC, the execution time for HOODG varies from about 3 seconds for Monk 1 and Parity5+5, to 20 minutes for the vote database. The large time requirement in the vote database is due to the large correlations between the attributes, which make many attributes weakly relevant (John, Kohavi, & Pfleger 1994), thus forcing a two-ply lookahead.

HOODG does much better on all the artificial data sets, and about the same on the real datasets, except for breast-cancer in the original encoding, where we noted that its myopic hill-climbing forces a ten-way split at the root variable. This is circumvented in the binary encoding because the equivalent of a multi-way split requires extra non-constant intermediate nodes, thus penalizing such a split[2].

We now turn to further empirical experiments to help us evaluate the appropriateness of the algorithm and the structure. We picked two artificial domains on which to conduct further experiments. The first was the Monk 1 problem mentioned in the previous section. The second was the multiplexer problem. In the multiplexer problem, there are $n$ "address bits" and $2^n$ "data" bits. An instance is labelled positive if the data bit indicated by the address bits is on. The structure of the smallest target concept is a tree, even if the concept space allows general graphs. The $n$ address bits are tested first, and then all the (different) data bits are tested on the same level. There is no advantage in trying to construct a graph or OODG; moreover, one would expect the oblivious restriction to make the task harder for HOODG, since each data bit would need to be tested on a different level. As noted in (Quinlan 1988), this domain is hard for decision trees also, since the entropy criteria favors a data bit at the root. The learning curves in Figure 3 show the accuracy of the hypotheses versus the training set size for HOODG and ID3. Each data point in the graph is the average of 10 runs on uniformly sampled training sets for that size. The twenty training set sizes were chosen at increments equal to 5% of the instance space.

The graphs clearly show that for Monk 1, HOODG has a large advantage over ID3, mainly because the concept is graph-like. HOODG quickly reaches the 100% level, while ID3 trails behind. Only at 410 instances, do all 10 runs of ID3 yield an accuracy of 100%. In the multiplexer domain, the algorithms perform roughly the same (within one standard deviation except for one data point at 42).

## Summary and Future Work

We have described OODG, a structure for representing concepts, and a hill-climbing algorithm, HOODG, for inferring OODGs from labelled instances. Tie-breaking heuristics were added to the original HOODG algorithm, improving the accuracy and learning rate.

Although limited in its myopic view, the HOODG algorithm performs well, especially if the concept is graph-like (*e.g.*, Monk 1, Monk 2), or has totally irrelevant attributes (John, Kohavi, & Pfleger 1994). The algorithm clearly outperforms other algorithms on the artificial domains tested, and is comparable on real domains, even though it currently does not deal with noise and probably overfits the data. Theorem 1 shows that the greedy approach always creates an OODG that is locally optimal if the full instance space is given. Theorem 2 shows that a multi-level projection step that finds the minimal width is intractable in the worst case (unless P=NP).

Deeper lookahead for variable selection is an obvious possible extension, especially since one motivation for growing the graph from the bottom is the asymmetric bound on the width at the different levels.

The OODG structure can be extended to allow splits on ranges and continuous values, but more research is

[2] The multi-way split problem suggests using a different measure, perhaps similar to Quinlan's gain-ratio (Quinlan 1986).

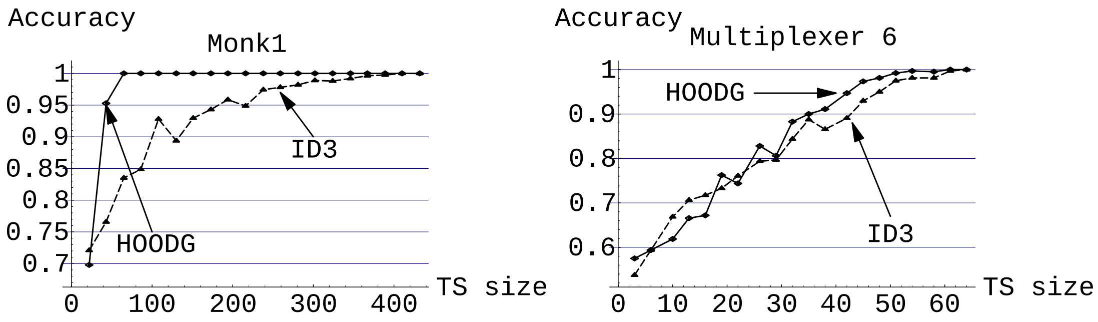


Figure 3: Learning curves for HOODG and ID3 and Monk 1 and 6 multiplexer.

required to extend the HOODG algorithm itself. Other important issues include dealing with noise (pruning), handling unknown values (here graphs might have advantages over trees due to reconverging paths), and finding more clever methods for ordering the variables

**Acknowledgements** We would like to thank the anonymous reviewers, James Kittock, Andrew Kosoresow, Shaul Markovitch, Nils Nilsson, Karl Pfleger, Eddie Schwalb, Yoav Shoham, and Tomas Uribe for their comments. The experimental section would not have been possible without the $\mathcal{MLC}$++ project, partly supported by National Science Foundation Grant IRI-9116399. We wish to thank everyone working on $\mathcal{MLC}$++, especially Richard Long.

## References

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

Burch, J. R.; Clarke, E. M.; and Long, D. E. 1991. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 403–407.

Friedman, S. J., and Suppowit, K. J. 1990. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers* 39(5):710–713.

Fujita, M.; Matsunaga, Y.; and Kakuda, T. 1991. On variable ordering of binary decision diagrams for the application of multilevel logic synthesis. In *Proceedings of the European Conference on Design Automation*, 50–54. IEEE Computing Press.

John, G.; Kohavi, R.; and Pfleger, K. 1994. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, 121–129. Morgan Kaufmann. Available by anonymous ftp from:
`starry.Stanford.EDU:pub/ronnyk/ml94.ps`.

Kohavi, R. 1994. Bottom-up induction of oblivious, read-once decision graphs. In *Proceedings of the European Conference on Machine Learning*. Available by anonymous ftp from
`starry.Stanford.EDU:pub/ronnyk/euroML94.ps`.

Lund, C., and Yannakakis, M. 1993. On the hardness of approximating minimization problems. In *ACM Symposium on Theory of Computing*.

Moret, B. M. E. 1982. Decision trees and diagrams. *ACM Computing Surveys* 14(4):593–623.

Oliver, J. J. 1993. Decision graphs — an extension of decision trees. In *Proceedings of the fourth International workshop on Artificial Intelligence and Statistics*, 343–350.

Pagallo, G., and Haussler, D. 1990. Boolean feature discovery in empirical learning. *Machine Learning* 5:71–99.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1:81–106. Reprinted in Shavlik and Dietterich (eds.) Readings in Machine Learning.

Quinlan, J. R. 1988. An empirical comparison of genetic and decision-tree classifiers. In *Proceedings of the Fifth International Conference on Machine Learning*, 135–141. Morgan Kaufmann.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Los Altos, California: Morgan Kaufmann.

Takenaga, Y., and Yajima, S. 1993. NP-completeness of minimum binary decision diagram identification. Technical Report COMP 92-99, IEICE.

Thrun, S.; Bala, J.; Bloedorn, E.; Bratko, I.; Cestnik, B.; Cheng, J.; De Jong, K.; Dzeroski, S.; Fahlman, S.; Fisher, D.; Hamann, R.; Kaufman, K.; Keller, S.; Kononenko, I.; Kreuziger, J.; Michalski, R.; Mitchell, T.; Pachowicz, P.; Reich, Y.; Vafaie, H.; de Weldel, W. V.; Wenzel, W.; Wnek, J.; and Zhang, J. 1991. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.