

Decision Graphs — An Extension of Decision Trees

JONATHAN J. OLIVER
Department of Computer Science
Monash University
Clayton, Victoria, 3168, AUSTRALIA

(jono@molly.cs.monash.edu.au)

Abstract: In this paper, we examine *Decision Graphs*, a generalization of decision trees. We present an inference scheme to construct decision graphs using the Minimum Message Length Principle. Empirical tests demonstrate that this scheme compares favourably with other decision tree inference schemes. This work provides a metric for comparing the relative merit of the decision tree and decision graph formalisms for a particular domain.

1 Introduction

In this paper, we examine the problem of inferring a decision procedure from a set of examples. We examine the decision graph [5, 1, 16, 15, 14], a generalization of the decision tree [3, 18], and propose a method to construct decision graphs based upon Wallace’s Minimum Message Length Principle (MMLP) [24, 10, 25]. The MMLP is related to Rissanen’s Minimum Description Length Principle (MDLP) [21, 22, 20]. For the reader unfamiliar with minimum encoding methods (MML and MDL), a good introduction to the area is given by Georgeff [10].

We formalize the problem of inferring a decision procedure from a set of examples as follows. The given data represents a set of objects. Each object is described in terms of the independent variables or “attributes”. Each object has a class or “dependent variable” associated with it. The data consists of vectors that give values for the attributes and class of each object.

The object of supervised learning is to find, within a specified family of functions, the function of the attributes that best predicts the class of an object. We will consider the given data as a *training set* and from this data determine the function that has the highest chance of accurately predicting the class of new objects. The functions we wish to consider are those that partition the set of objects into categories, such that, as nearly as possible, all objects in a category have the same class.

In the next section, we present an overview of decision trees and then examine the decision graph. We argue that some decision functions are more simply described by decision graphs than decision trees. In Section 2, we discuss the Minimum Description Length Principle and present a hill climbing inductive inference algorithm based on the MMLP. In Section 3, we adapt the hill climbing algorithm to design an algorithm to construct decision graphs. In Section 4, we present some results comparing the decision graph scheme with other decision tree inference schemes. In Section 5, we describe how we can decide whether a decision tree or a decision graph is more appropriate for a particular domain.

1.1 An Overview of Decision Trees

A decision tree for the proposition $(A \wedge B) \vee (C \wedge D)$ is depicted in Figure 1. It consists of *decision nodes* [Drawn as ovals in Figure 1] and *leaves* [Drawn as rectangles in Figure 1]. Decision nodes specify an attribute to test upon an object, with the arcs out of the decision node specifying the possible values that attribute can take. Each leaf of the decision tree specifies a category. A decision tree with c leaves partitions the space of objects into c disjoint categories.

Any object is categorized by determining the unique leaf associated with that object. The leaf associated with an object is defined by following the path down the tree such that the arc out of each decision node has the same value as that object’s value for that attribute.

Each leaf of a decision tree is labelled with a *default class*, typically the most frequent class of the example objects associated with that leaf. A new object is classified by determining the leaf it is associated with, and classifying it as the default class of that leaf.

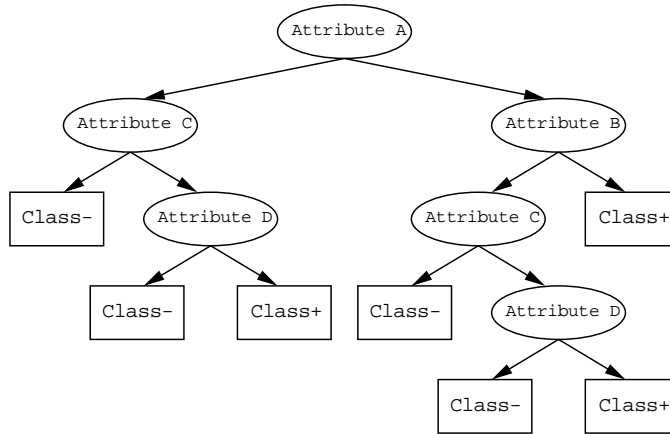


Figure 1: A decision tree for $(A \wedge B) \vee (C \wedge D)$

1.2 The Replication Problem

The decision tree shown in Figure 1 gives an inefficient representation of the proposition

$$(A \wedge B) \vee (C \wedge D)$$

While the term $(A \wedge B)$ is described efficiently, the term $(C \wedge D)$ requires two identical subtrees to be represented. The reason for this duplication is the disjunction in the proposition. In general, conjunctions can be described efficiently by decision trees, while disjunctions require a large tree to describe. This representational shortcoming of decision trees has been identified by [Pagallo and Haussler] [17] and [Mathues and Rendell] [13], and has been termed the *replication problem*. [Pagallo and Haussler] demonstrate that many boolean functions with small DNF (disjunctive normal form) descriptions will be inefficiently described by decision trees [17].

The effect of the replication problem is that decision tree learning algorithms require a large amount of data to learn disjunctive functions. For example, if a decision tree learning algorithm requires D data items to learn the subterm $(C \wedge D)$ (in Figure 1), then the same algorithm will require at least $2 \times D$ data items to learn the function $(A \wedge B) \vee (C \wedge D)$. This follows since there are two replications of the subtree which describes $(C \wedge D)$ ¹.

One solution to the replication problem is to allow decision nodes to contain features that are a function of one or more attributes. A *feature* consists of a conjunction of literals (where each *literal* is an attribute or a negated attribute).

[Pagallo and Haussler] [17] and [Mathues and Rendell] [13] construct decision trees, and analyse the tree to determine likely candidate features. They then add these features to the list of attributes and build another tree. Both groups repeat this process until no more features can be added. There is a need to extend these results for those cases when the data has significant noise, and when the data has more than two classes.

Other attempts to resolve the replication problem include partitioning an attribute's values (described in Section 1.3) and allowing a more expressive structure, namely decision graphs (described in Section 1.4).

1.3 The Fragmentation Problem

Another aspect of the replication problem can occur when the data contains attributes with more than 2 values. If a tree uses high arity attributes (say arity ≥ 10), then it will quickly fragment the data into many partitions (see Figure 2).

The most commonly used method to avoid fragmentation of the training data is to construct subsets of the attribute's values (see Figure 3) [3, 11, 6, 1, 9].

¹A similar argument applies if the subterm $(C \wedge D)$ appears higher in the tree than $(A \wedge B)$.

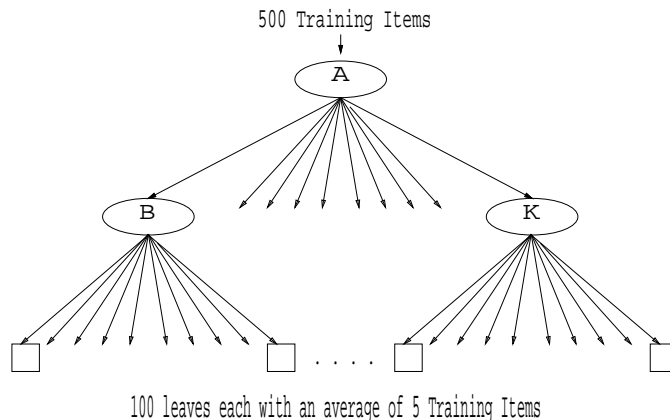


Figure 2: A decision tree demonstrating the fragmentation problem

It has been noted [3, 18] that searching through the possible 2-way subsets is a computationally expensive operation. If an attribute has arity M , then there will be $2^{M-1} - 1$ non-trivial 2-way subsets to explore. This approach is therefore infeasible for dealing with high arity attributes. For example, with a problem such as protein secondary structure prediction, where an amino acid has 20 possible values, there would be $2^{19} - 1 = 524,287$ 2-way subsets to explore.

[Breiman et. al.] [3] presented a linear time algorithm to find a locally optimal binary partition, when there are only 2 classes.

[Chou] [6, 5] presented a fast clustering algorithm that finds locally optimal N-way subsets. However, this method does not offer a method for selection between different sized partitions. For example, it does not offer a metric for choosing between a 2-way and a 3-way subsetting of an attribute's values (see Figure 3).

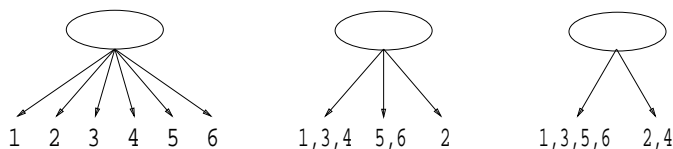


Figure 3: A 6-way, 3-way and 2-way partitioning of an attribute's values

1.4 The Decision Graph

In this paper, we examine what we consider an elegant solution to the replication and fragmentation problems. This solution involves merging duplicated subtree with a *Join* operator as shown in Figure 4. *Decision Graphs*, such as the one depicted in Figure 4, are generalizations of decision trees, having decision nodes and leaves. The feature that distinguishes decision graphs from decision trees is that decision graphs may also contain Joins. A Join is represented by two nodes having a common child, and this specifies that two subsets have some common properties, and hence can be considered as one subset. The manner in which objects are categorized by decision graphs is the same as that of decision trees.

Each decision tree and decision graph define a *categorization* (i.e., some partitioning of the object space into disjoint categories). The set of decision functions representable by graphs is exactly the same as the set representable by trees. However, the set of categorizations which can enter into the definition of a decision function are different. For example, the categorizations for

$$(A \wedge B) \vee (C \wedge D)$$

given in Figure 1 and Figure 4 are different since the decision tree partitions the object space into 7 categories, while the decision graph partitions the object space into 2 categories.

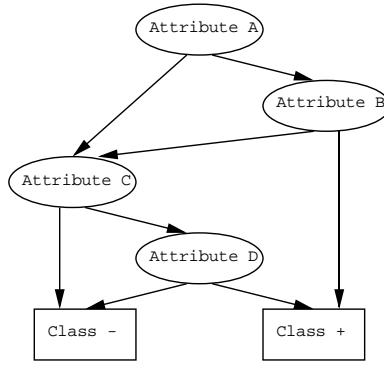


Figure 4: A decision graph for $(A \wedge B) \vee (C \wedge D)$

Decision graphs have been previously examined in the literature by [Chou] [5], [Bahl et. al.] [1] and [Mahoney and Mooney] [12].

[Mahoney and Mooney] used the following method to construct decision graphs:

1. Grow a decision tree.
2. Search the decision tree for related subtrees.
3. Merge the related subtrees to build a decision graph.

They reported limited success in doing this, and concluded that “searching for similar subtrees that need to be collapsed into a shared structure is combinatorially explosive”.

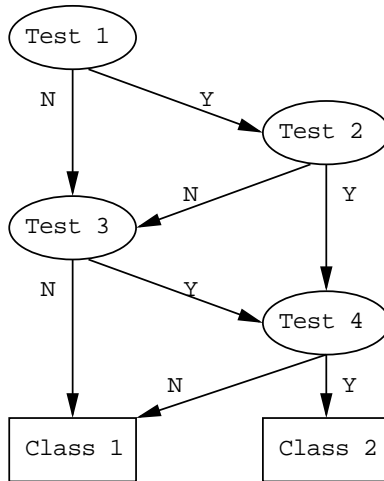


Figure 5: A Decision Pylon

[Chou] and [Bahl et. al.] approached the construction of decision graphs using a different method to [Mahoney and Mooney]. Both groups determined the *gross structure* of the decision graph a priori. For example, the gross structure of a decision pylon ([Bahl et. al.] termed their structures *decision pylons*) was determined a priori to look like a pylon, as shown in Figure 5.

Both groups applied partitioning algorithms (discussed in Section 1.3) if a split took the structure of the graph being constructed outside the predetermined structure.

Fixing the structure of a decision graph a priori is a considerable disadvantage [7], since often the shape of the decision graph provides insight into the domain under consideration. The decision graphs presented in Section 4 highlight this disadvantage.

The contribution of this paper is that it applies a minimum encoding technique (MML) to the construction of decision graphs. The method described in this paper avoids the problems encountered by [Mahoney and Mooney] [12] since it does not search through decision trees for related subtrees. Furthermore, the method presented here obviates the need for someone to arbitrarily predetermine the gross structure of a decision graph before the graph is grown, the gross structure is determined by the data.

2 The Minimum Message Length Principle

In the introduction to this paper, we stated the intention to present an algorithm to grow decision graphs based on the Minimum Message Length Principle (MMLP). Minimum Message Length is a criterion for comparing inductively-derived theories and the explanations that these theories provide.

We shall begin by going over the theory behind the Minimum Message Length Principle, and then give an overview of how the MMLP can be used to design algorithms for inductive learning. The MMLP does not prescribe any general algorithm for inductive inference. It provides a metric which allows us to search through a model space for acceptable theories. In Section 2.2 we present a hill climbing algorithm that iteratively modifies and extends a theory. In Section 3 we apply the hill climbing algorithm to the problem of inferring decision graphs and present an algorithm that grows decision graphs.

2.1 The Induction Principle

The output of a program that performs inductive inference (whether the output is a decision tree, probabilistic finite state automaton, class structure, or an estimator for some parameter) can be regarded as a theory about the input data.

We identify some subset of the input data, D , as being explained by a theory, T . This subset of the input data is dependent upon the learning paradigm we wish to use. For example, when performing supervised learning the theory will explain the class of each example, when performing classification the theory will explain the attribute values for each example, and when inferring probabilistic finite state automata the theory will explain the sequence of symbols in the input data.

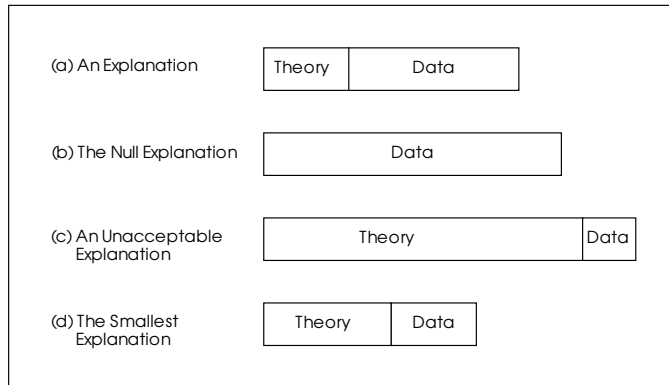


Figure 6: A Range of Explanations

We define an *explanation* of this subset, D , as a message. This message will be encoded as a binary string and consists of two parts as shown in Figure 6(a). The first part states a theory, T , about D , and the second part states D in full using a code that gives the minimum expected message length if T is true.

By standard coding theory, the length of the second part of the message is:

$$-\log_2 \text{Prob}(D|T)$$

If we assume a prior probability distribution over the set of theories, then we can encode the first part of the message with length:

$$-\log_2 \text{Prob}(T)$$

However, there may be no obvious probability distribution to assign to complex theories, such as decision trees. In these cases, we must design a code to express the theory as a binary string. However, the use of a non-redundant code in fact implies a prior probability distribution over theories.

The total length of an explanation is therefore:

$$-\log_2 \text{Prob}(T) - \log_2 \text{Prob}(D|T)$$

The null explanation (shown in Figure 6(b)) is a message describing the data using a null theory. A theory is *unacceptable* (shown in Figure 6(c)) if its explanation of the data is longer than the null explanation. The shortest explanation (shown in Figure 6(d)) of T and D is considered the best explanation. If the data is totally random (in the sense of Chaitin [4]) then the best theory will be the null theory.

2.2 Comparing Theories

When given two theories, T_1 and T_2 , about some data we may calculate the message length of the explanations provided by T_1 and T_2 . We may then say that one theory explains the data better than the other theory. When we do this it is not necessary to actually encode the theory and data: all we need to do is calculate the length of the explanation.

If there is a finite number of possible theories, then the MMLP can be used to find the “best” theory, T_{best} . This is done by calculating the length of the explanation provided by every theory and the theory with the shortest explanation, T_{best} , is considered the best theory. However, it is not normally feasible to consider every possible theory. In some circumstances, such as with classification, there are an infinite number of theories. In other circumstances, such as decision tree inference, while there are a finite number of theories, there are too many theories to explore in a reasonable amount of time.

2.2.1 Hill Climbing Algorithm

To overcome this problem, we may use the MMLP to help us search for a suitable theory. Since the MMLP allows us to compare two theories and say one is superior to the other, we may begin with some theory, and iteratively propose improvements to this theory until no more improvements can be found, as shown in Figure 7.

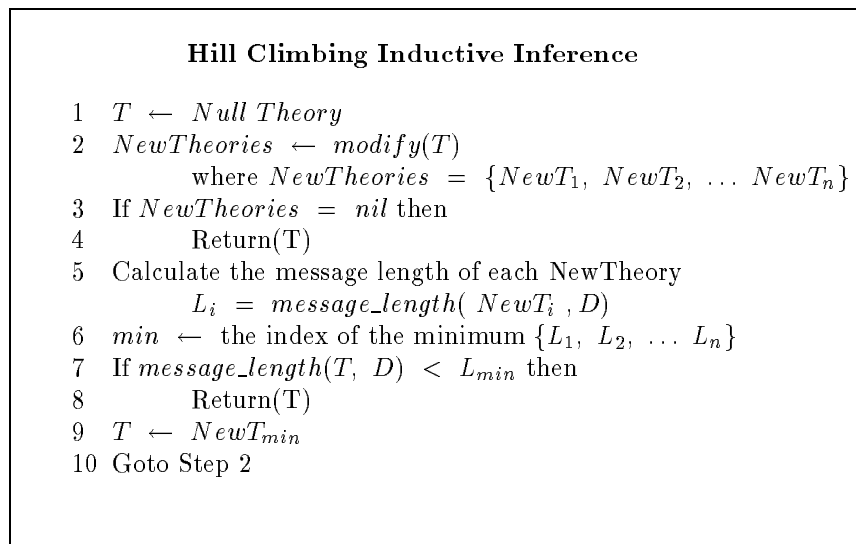


Figure 7: An MML Hill Climbing algorithm

The algorithm presented in Figure 7 requires us to define two procedures:

- $\text{modify}(T)$ constructs a list of modifications to T .
- $\text{message_length}(T, D)$ returns the length of the message when T is used to explain D .

2.2.2 The message_length procedure

In this paper, we will assume the existence of message_length procedures for the theories under consideration (decision trees and decision graphs). Procedures for encoding decision trees have been described in [Wallace and Patrick] [25] and [Quinlan and Rivest] [20]. A procedure for encoding decision graphs has been described in [Oliver and Wallace] [16].

2.2.3 Lookahead

The hill climbing algorithm proposed in Figure 7 only considers one step modifications. Thus, the algorithm may get stuck in a local minimum, and not find the global minimum. To overcome this, the modify procedure may be designed with an additional lookahead parameter that specifies number of base level modifications we are willing to consider at any one stage.

3 Constructing Decision Graphs

In this section, we will present an algorithm for inferring decision trees using the MMLP proposed by [Quinlan and Rivest][20]. We will examine why this algorithm is inappropriate for the construction of decision graphs. In Section 3.2 we will examine differences between the inference of decision trees and the inference of decision graphs. We go on in Section 3.3 to adapt the Hill Climbing Algorithm given in Figure 7 to design a decision graph inference algorithm.

3.1 An Overview of a MMLP Decision Tree Generation Algorithm

[Quinlan and Rivest] [20] proposed an algorithm for the generation of decision trees that consists of two phases, a *Growing Phase* and a *Pruning Phase*.

The Growing Phase begins with the root of the decision tree being a leaf with all of the training set associated with it. It extends the decision tree by iteratively performing the following procedure:

1. Let L be a leaf such that it is possible to replace L with a decision node.
2. For each attribute A that might be specified in the decision node to replace L , compute the communication cost if this change is made.
3. Replace L with the decision node of minimum communication cost.

The Pruning Phase consists of repeatedly replacing decision nodes (all of whose children are leaves) by leaves, whenever this improves the total communication cost.

3.2 Requirements of a Decision Graph Inference Scheme

There are two considerations that make the approach used by [Quinlan and Rivest 1989] unsuitable for the construction of decision graphs.

- Firstly, for a given leaf, L , it is now necessary to decide whether to (a) replace L with a decision node, or to (b) introduce a Join for L .
- Secondly, it does not attempt to provide the order in which a decision tree should be grown, since the order in which we grow a decision tree is immaterial. However, the order in which leaves are expanded when growing a decision graph affects the resultant graph.

We propose the following algorithm to develop decision graphs which addresses the above considerations. It uses the MMLP to (a) determine for each leaf whether to Split that leaf, or to introduce a Join for that leaf, and (b) choose which leaf should be expanded (whether with a Split or a Join).

3.3 A Decision Graph Generation Algorithm

We begin with the root of the graph being a leaf. We extend the graph by iteratively performing the procedure Grow Graph until the graph is perfect or cannot be grown any further.

Grow Graph

1. For each leaf, L , determine the attribute A which it should be split on. Record, but do not perform, the alteration (Split L on A) along with its saving in `message_length`.
2. For each pair of leaves, L_1 and L_2 , perform a tentative Join. Record, but do not perform, the alteration (Join L_1 and L_2) along with its saving in `message_length`.
3. Choose the alteration (whether from step 1 — a Split, or from step 2 — a Join) that has the greatest saving. If this alteration creates a savings in `message_length`, then perform that alteration to the graph.

3.4 Hurdling

We note that procedure Grow Graph does no *hurdling* operations, i.e., performing alterations that increase the `message_length`. We stipulate this so the process will terminate. If Grow Graph performed a hurdling operation that split a node, N , into N_1 and N_2 then the next iteration of Grow Graph would Join N_1 and N_2 back into N (since joining them back together would save `message_length`).

4 Test Results

Data Set	Decision Graph	Wallace and Patrick	Quinlan and Rivest	C4
Hypo	0.6%	0.6%	0.6%	0.55%
Discordant	1.1%	1.1%	1.9%	1.25%
LED	26.5%	26.5%	26.9%	28.1 %
Endgame	15.2%	15.2%	17.9%	13.6 %
xd6	10.0%	15.0%	20.5%	14.9 %

Table 1: The error rates of a variety of methods on standard data sets

In order to test the feasibility of using the Minimum Message Length Principle to generate decision graphs, we implemented it and compared its results with the results produced by decision tree inference schemes. We compared it with C4 with Pessimistic Pruning [19], and with two tree generation schemes that use the Minimum Message Length Principle, one by [Quinlan and Rivest] [20] and the other by [Wallace and Patrick] [25].

Results are given for five data sets “Hypo”, “Discordant”, “LED”, “Endgame” and “xd6” in Table 1. These data sets are described in [Quinlan] [19].

4.1 Learning Disjunctive Concepts

When we tested our algorithm upon the “xd6” data set we found the Join operator was used extensively. In fact, the “xd6” data set is an artificial set with 10 attributes that was generated so that a division into categories according to the Boolean function of attributes 1 to 9:

$$a_1 \wedge a_2 \wedge a_3 \vee a_4 \wedge a_5 \wedge a_6 \vee a_7 \wedge a_8 \wedge a_9$$

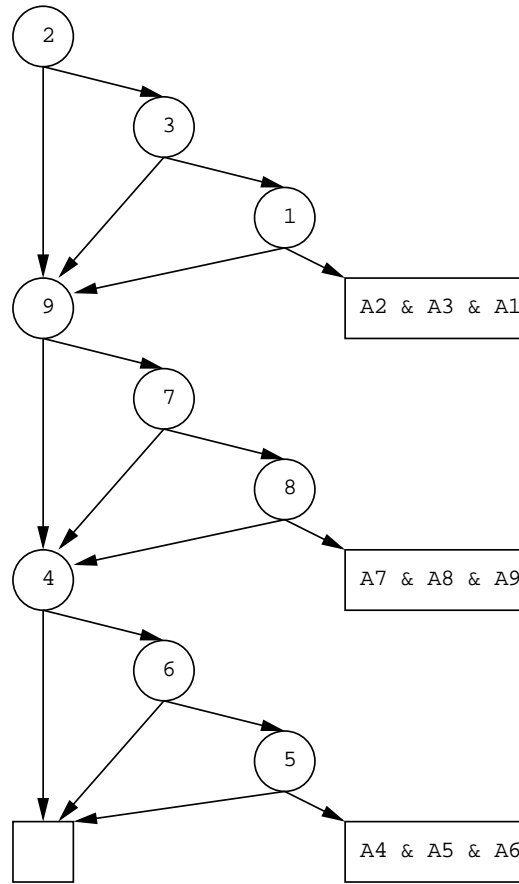


Figure 8: A decision graph for the xd6 data set

should give categories with 90% purity. The smallest possible decision tree for this function has 39 decision nodes and 40 leaves. This function is hard for decision tree inference schemes to learn, since the data is partitioned into 40 subsets. The graph inference scheme re-constructed this function, as shown in Figure 8, and hence the error rate of the decision graph is the noise level of 10%.

Number of Data Items	Decision Nodes	Leaves	Error Rate
400	7	8	22.4%
1000	24	25	13%
2000	33	34	10.6%
2500	33	34	10.2%
2800	34	35	10%
3000	39	40	10%

Table 2: Varying the size of the xd6 data set — Decision Tree

Number of Data Items	Decision and Join Nodes	Leaves	Error Rate
300	14	6	19%
350	17	6	17%
400	21	4	10%

Table 3: Varying the size of the xd6 data set — Decision Graph

In Table 2 and Table 3 we give results when we varied the number of items in the training set of the “xd6” data set. The decision graph inference scheme re-constructed the original function from 400 data items, where [Wallace and Patrick’s] decision tree inference scheme required 3000 data items to reconstruct the function. Furthermore, the decision graph produced a decision procedure that was easier to identify from a human’s perspective.

We had similar success with the 1st Monk’s data set (reported in [23]). This data set is an artificial data set constructed from the function:

$$Jacket_Color = Red \vee Head_Shape = Body_Shape$$

While ID3 had an error rate of 31% [23], the decision graph inference scheme reconstructed the original function, as shown in Figure 9, and hence had a 0% error rate.

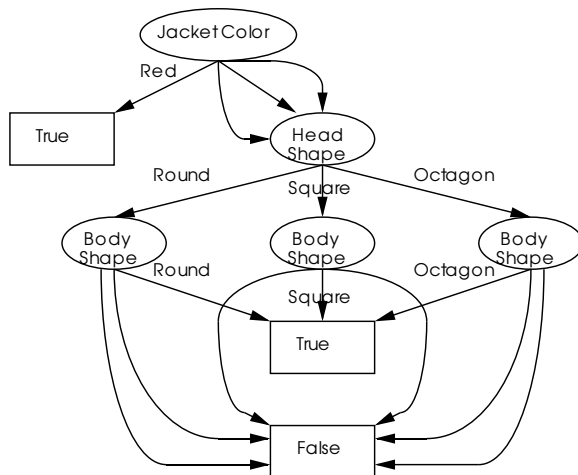


Figure 9: A decision graph for the 1st Monk’s data set

4.2 Protein Data Set

We tested decision graphs on some data sets generated from a protein structure database [8]. The data consisted of amino acid chains with a secondary structure specified at each point. Microbiologists can determine the amino acid chain of a protein, but finding the secondary structure ($\{ Extended, Helix, Other \}$) which is related to the shape of the protein is quite difficult. We constructed decision graphs that predicted the secondary structure at a point in a protein by using a window (centered at the point of interest) of the amino acid chain as attributes.

Figure 10 shows an abbreviated form of decision graph generated when a window of the amino acid chain was used for attributes. In this example, the arity of the 7 attributes is 20, and hence decision tree inference schemes would quickly fragment the data into small sets where learning would be difficult. For example, a complete height 3 tree would partition the data into 8000 subsets. Each leaf of a height 3 tree would have little data associated with it, and the tree would be unintelligible to biologists.

The decision graph in Figure 10 displays splits where subsets of the amino acids’ values have been formed. For example, the root of the graph splits the data into 4 subsets:

$$\{ A, E, K, Q \} \quad \{ F, I, L, V \} \quad \{ C, D, G, H, M, N, R, S, T, W, Y \} \text{ and } \{ P \}$$

The subsets of the amino acids’ values were found, not by an exhaustive search, but by successive joins. After the first split (on attribute 4 the central amino acid of the window) the graph had 20 leaves. The Grow Graph procedure explored the possibility of each leaf joining with another leaf, which in the case of 20 leaves gives 190 cases to explore. In general, when there are N leaves, the Grow Graph procedure will explore $\frac{N \times (N-1)}{2}$ possible joins.

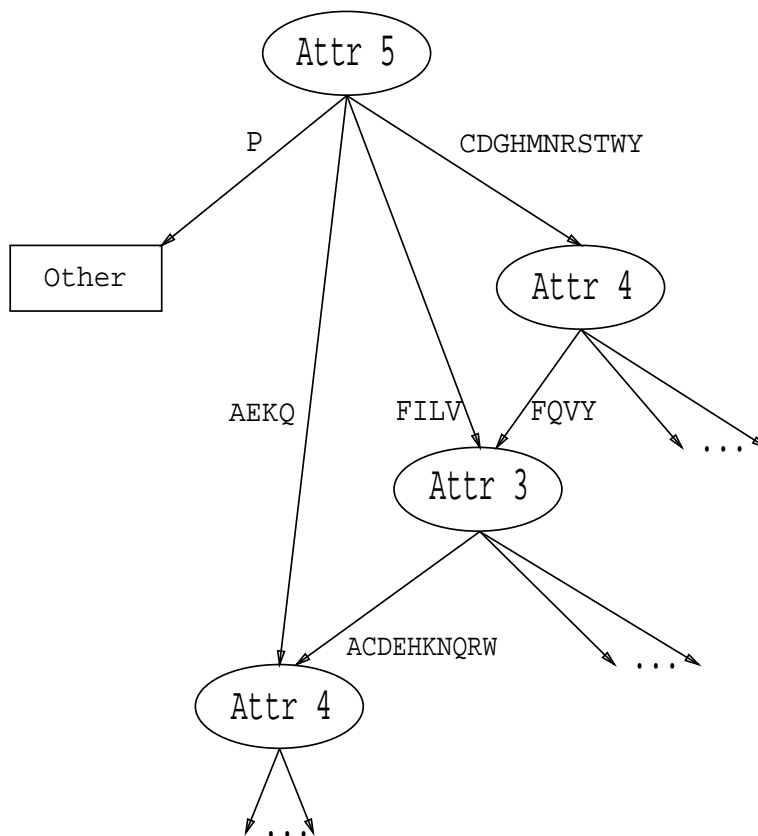


Figure 10: A abbreviated decision graph for the protein structure data set

5 Determining the Type of Model

The reader will note that the decision graph scheme often gave results identical to the decision tree scheme proposed by [Wallace and Patrick] [25]. In these cases, the decision graph scheme used the MML metric to determine that a decision tree was more appropriate for this data set than a decision graph. In this section, we describe how this determination was made.

The message_length function described in [Oliver and Wallace] [16] has a parameter, P_J the probability of a Join, which describes how disjunctive a decision graph is. P_J effects the message_length of a given decision graph. If P_J is high (for example, 0.50) then Joins cost a small number of bits (2 bits per binary Join). However, if P_J is low (for example, 0.10) then a Join is relatively expensive (4.6 bits per binary Join), and if $P_J = 0$ then the cost of a Join is infinite. Hence, procedure *Grow Graph* will grow a tree if $P_J = 0$, and it will grow graphs with more Joins as P_J is increased.

P_J	Message Length
0.00	84.50 bits
0.10	76.26 bits
0.20	47.04 bits
0.30	47.63 bits
0.40	49.50 bits
0.50	52.68 bits

Table 4: 1st Monks Data Set — Null Explanation 129.75 bits

In Tables 4 and 5 we vary the P_J parameter for 2 data sets. We select that value of P_J that minimizes the message_length. We found that the 1st Monks data set was disjunctive in nature, and hence decision trees were inappropriate for this data set. The decision tree made an excellent explanation for the Hypothyroid

P_j	Message Length
0.00	202.4 bits
0.10	204.3 bits
0.20	206.6 bits
0.30	209.3 bits
0.40	212.6 bits
0.50	217.1 bits

Table 5: Hypothyroid Data Set — Null Explanation 1786.4 bits

data set (saving 1584 bits), which was better than any decision graph explanation found.

6 Conclusion

The MMLP gives us a measure to compare theories with distinct structures, and allows us to determine if one theory is superior to another. Within the scope of decision graphs and decision trees, using a minimum encoding framework:

- allows us to compare modifications that are totally different in nature. For example, if we have a decision graph G , and we postulate two modifications to G , one which splits a node, the other which joins two nodes, then we can sensibly compare these modifications and say one modification is superior to another.
- provides a theoretical guarantee. [Barron and Cover] [2] show that if the data is drawn from a population in which the probability distribution over classes can be exactly represented by a decision tree or decision graph function then, given sufficient data, an MML tree or graph inference algorithm is guaranteed to recover that function.

Representing the theory as a directed acyclic graph rather than a tree resulted in a more expressive language for theories. The additional expressiveness results in a search through a larger search space, however, the benefits can be significant.

- Firstly, the graph formalism describes a large set of disjunctive functions more simply than the tree formalism. Hence, the decision graph algorithm will in general infer functions in this set with fewer data than would be required by a decision tree algorithm.
- Secondly, the graph formalism can use attributes with many values, without fragmenting the data into small partitions. The resultant graphs form subsets of attribute values that be of interest to domain experts.
- Thirdly, the applicability of decision graphs for a particular domain can be determined. This is done by comparing the message lengths of the best decision graph with that of the best decision tree for that domain.

7 Acknowledgments

I would like to thank my supervisor Ingrid Zukerman for valuable advice, Chris Wallace, David Dowe and Wray Buntine for valuable discussions, and Phil Chou for critical reading of the manuscript.

References

- [1] L.R. Bahl, P.F. Brown, P.V. deSouza, and R.L. Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:1001–1008, 1989.
- [2] A.R. Barron and T.M. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, 37:1034–1054, 1991.

- [3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
- [4] G.J. Chaitin. On the length of programs for computing finite sequences. *J.A.C.M.*, 13:547–549, 1966.
- [5] P.A. Chou. *Applications of Information Theory to Pattern Recognition and the Design of Decision Trees and Trellises*. PhD thesis, Department of Electrical Engineering, Stanford University, June 1988.
- [6] P.A. Chou. Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4), 1991.
- [7] P.A. Chou. *Personal Communication*. 1992.
- [8] D.L. Dowe, J.J. Oliver, L. Allison, C.S. Wallace, and T.I. Dix. A decision graph explanation of protein secondary structure prediction. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS), Biotechnology Computing Track*, pages 669–678, 1993.
- [9] U.M. Fayyad and K.B. Irani. The attribute selection problem in decision tree generation. In *Proceedings of AAAI-92*, pages 104–110, 1992.
- [10] M.P. Georgeff and C.S. Wallace. A general criterion for inductive inference. In *Proceedings of the 6th European Conference on Artificial Intelligence*, 1984.
- [11] I. Kononenko, I. Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.
- [12] J.J. Mahoney and R.J. Mooney. Initializing ID5R with a domain theory: Some negative results. 91-154, Department of Computer Science, University of Texas at Austin, Taylor Hall 2.124, Austin, Texas 78712-1188, USA, March 1991.
- [13] C.J. Mathues and L.A. Rendell. Constructive induction on decision trees. In *IJCAI-89*, pages 645–650, 1989.
- [14] J.J. Oliver. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. More extensive version available as TR 173, Computer Science Department, Monash University, Vic 3168, AUSTRALIA.
- [15] J.J. Oliver, D.L. Dowe, and C.S. Wallace. Inferring decision graphs using the minimum message length principle. In A. Adams and L. Sterling, editors, *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 361–367. World Scientific, Singapore, 1992.
- [16] J.J. Oliver and C.S. Wallace. Inferring decision graphs. In *Proceedings of Workshop 8 — Evaluating and Changing Representation in Machine Learning IJCAI-91*, 1991. Also available as TR 170, Computer Science Department, Monash University, Vic 3168, AUSTRALIA.
- [17] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.
- [18] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [19] J.R. Quinlan. Simplifying decision trees. *International Journal of Man Machine Studies*, pages 221–234, 1987.
- [20] J.R. Quinlan and R.L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.
- [21] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11:416–431, 1983.
- [22] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore, 1989.
- [23] S.B. Thrun et al. The monk’s problems: A performance comparison of different learning algorithms. CMU-CS-91-197, Carnegie Mellon University, December 1991.
- [24] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, 11:185–194, 1968.
- [25] C.S. Wallace and J.D. Patrick. Coding decision trees. *To appear in Machine Learning*, 1993. Also available as TR 153, Computer Science Department, Monash University, Vic 3168, AUSTRALIA.