Very Fast Decision Table Execution of Propositional Expert Systems

Robert M. Colomb
Charles Y.C. Chung

CSIRO Division of Information Technology
Box 1599 North Ryde NSW 2113 Australia
colomb@syd.dit.csiro.au

Abstract

A formal equivalence between propositional expert systems and decision tables is proved, and a practicable procedure given to perform the transformation between propositional expert systems and decision tables. The method gave an order of magnitude speed increase for a well-known expert system in routine use. The method is very general: adaptations are shown for forward and backward chaining inferencing engines, inexact reasoning, and systems where some facts have a high cost and must be determined only if necessary. A particular application for the decision table representation is in real-time expert systems, since a simple hardware implementation is available which gives further orders of magnitude increase in performance. Finally, the decision table representation greatly simplifies the problem of completeness and consistency checking.

Introduction

Expert systems which rely on the propositional calculus are very common, and the standard implementations are very computationally expensive, both in time and memory. For example, the COLOSSUS system (Beinat & Smart 1989) has 6500 rules, and runs on a very large mainframe computer. It requires 20 megabytes of real memory per user, and has a response time measured in minutes.

In this paper, we show that a propositional expert system can be mechanically transformed into a decision table. Decision tables are very simple computational structures which can be executed very quickly and require little memory. With very simple hardware assist, it is possible to build systems with hundreds of rules with execution times of a few tens of microseconds, which could greatly expand the useful domain of expert system technology, especially in real time applications. In addition, the decision table representation greatly simplifies the problem of checking rules for completeness and consistency.

After a few definitions, we present a general proof of the equivalence of propositional expert systems and decision tables. We then describe a computationally practicable algorithm for performing the transformation, and describe its application to a real system. The algorithm can be easily adapted to a much more space efficient product, shown in the next section. We describe generalizations to inexact reasoning systems and systems where some facts have a high cost and must be obtained only if necessary. We show how the decision table form is particularly adapted to real time applications, and finally consider consistency and completeness checking in the decision table representation. The paper completes with a conclusion section.

This restriction to propositional systems differs from the main stream of parallel production system research (Gupta *et al.* 1986, Stolfo 1985), which concentrates on systems like OPS-5, which are based on the first order predicate calculus.

Definitions

A propositional expert system is a set of propositions in Horn clause form. Each clause consists of an antecedent, which is a conjunction of elementary propositions, and a consequent, consisting of a single elementary proposition. A clause will be called a *rule* in the following. Note that a proposition with disjunctions in its antecedent or conjunctions in its conclusion can be easily transformed into clausal form.

Elementary propositions can be classified into three mutually exclusive groups: *facts*, which appear only in antecedents; *conclusions*, which appear only as consequents; and *assertions*, which appear both in antecedents and consequents. We designate the set of facts as $F = \{f_i\}$, conclusions as $C = \{c_i\}$, assertions $A = \{a_i\}$ and rules as $R = \{r_i\}$.

It is convenient to consider a fact as an assignment to a variable of one of a small number of possible values. We thus obtain a set X of variables x_i each of which has a set of possible values V_i, and every fact is a proposition of the form $x_i = v$ for v in V_i. An *input* I is a conjunction of assignments to a subset of the variables. The assignment is *incomplete* if the subset is proper. (An incomplete input corresponds to the values of some of the variables being unknown.)

In applying an expert system R to an input I, we obtain a propositional system P consisting of the conjunction of I with the disjunction of the rules in R, and consider the conclusions C. A conclusion is *determined by* the input if it is a logical consequence of P; *inconsistent with* the input if its negation is a logical consequence of P; and *consistent with* the input if its negation is not a logical consequence of P. A conclusion consistent with an input may be determined by it, but not necessarily.

A *decision table* is a table with one column for each variable and an additional column for an action. Each row of the table contains an assignment of values to variables and an associated action. An incomplete assignment is equivalent to a *don't care* condition for the variables without values. A decision table is executed by presenting it with an assignment I. If any row is a subset of I, its action is executed.

Equivalence Between Propositional Expert Systems and Decision Tables

A decision table is a propositional expert system. The assignment in a row is a conjunction of propositions, which is the antecedent of a rule. The action of a row is a conclusion. A row is therefore equivalent to a rule, and a collection of rules is a propositional expert system by definition. Note that the propositional system has only facts and conclusions, but no assertions. Such a system will be called a *flat* expert system and is clearly equivalent to a decision table.

A propositional expert system is a decision table. Consider an expert system R with an input I. Associate with I the subset of conclusions determined from the propositional system P given by the conjunction of I with the disjunction of rules in R. Call this subset R(I). R(I) can always be computed since the propositional calculus is decidable. R can thereby be seen as a function mapping the set of assignments into the set of subsets of conclusions. The number of possible assignments is finite. The function can in principle be expressed in an extensional form by writing down each assignment I as a row in a table and appending to it the subset of conclusions R(I). This form is by definition a decision table.

This proof is constructive, but unfortunately not practicable, since the number of assignments is exponential in the number of variables. It is, however, general. In particular, it is independent of the inference engine used and of the details of any rules involving assertions.

A Practicable Transformation Algorithm

This section presents a practicable transformation algorithm, first by making some restrictive assumptions. It is then shown how the algorithm can be modified to remove many of the restrictions. The restrictive assumptions are that the system uses a forward chaining inference engine, and that no assertion is negated in either an antecedent or consequent of a rule.

We consider the rules r_i in the expert system R as nodes in a graph. An arc is drawn between r_i and r_j if there is an assertion a which appears as the consequent of r_i and in the antecedent of r_j. We assume that this graph is acyclic: no proposition is a consequent of a rule of which it is an antecedent, or more generally, no proposition is a consequent of a rule which has an antecedent necessarily logically dependent on it. This graph is essentially the dependency graph of Nguyen, *et al.* (1985).

We partition the set of rules into B, those all of whose antecedents are facts; K, those whose consequents are conclusions; and M, the others. (B and K are assumed disjoint. Any rules in B intersect K are flat by definition, so can be removed and added to the flat expert system produced by the algorithm after its completion.) A node r can be labelled by the maximum number of arcs between it and a member of B, as shown in Figure 1. B is the set with label 0, and M can be partitioned into M_1, M_2, etc. each M_i consisting of the nodes with label i. The label of K is defined as the *maximum depth of reasoning* of the system R. As shown in the figure, we also label each arc with the assertion from which the arc is derived and each assertion with the maximum label of a node with that assertion as a consequent. This labelling can be done in the course of verifying that the graph has no cycles using an algorithm successively removing nodes with no input arcs.

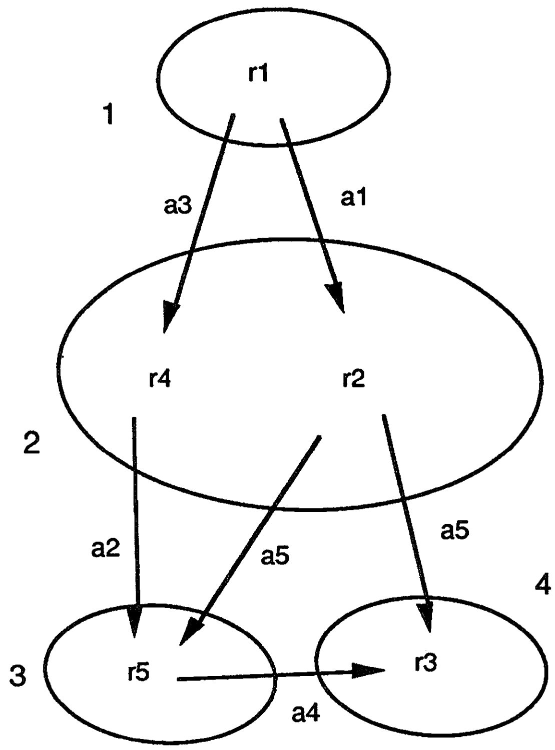

Figure 1
Graphical Representation of Rules Labelled by Distance From Base

Lemma 1: Every assertion in an antecedent of a rule of label i must have a label less than i. Proof follows directly from the definitions.

Lemma 2: There is at least one rule with every label up to the maximum depth of reasoning. Proof: if a level i were missing, then in the acyclic verification process when step i were reached there would be no nodes without input arcs and the graph would therefore be cyclic.

The algorithm proceeds by replacing each assertion a in the antecedents of rules labelled i with the disjunction of the antecedents of rules in labelled i-1 of which assertion a is a consequent, beginning with label 1. This process in effect collapses the rules onto the conclusions by successively replacing intermediate assertions with expressions which imply them. The resulting propositions have only facts in their antecedents and only conclusions as consequents, can be expressed in clausal form, therefore form a flat expert system and therefore are equivalent to a decision table.

A detailed presentation of the algorithm with a worked example is given in Colomb (1989).

The assumptions can be relaxed.

Analysis of Algorithm

The algorithm can be divided into two parts: construction of a labelled dependency graph, and production of the decision table from the labelled graph. We begin with a table of rules. An auxiliary data structure is required: an *alternatives count*, which will contain for each assertion the number of rules having that assertion as consequent. The dimensions of the problem will be:

r	the number of rules
a	the average number of assertions per rule
d	the maximum depth of reasoning

To construct the dependency graph, we first count the number of alternatives for each assertion. This requires one step per rule, therefore is O(r). Facts will be taken as having zero alternatives. We then proceed to identify the rules with label *1*, which are those rules all of whose antecedents have zero alternatives. When a rule is labelled, we decrement the number of alternatives for its consequent assertion, and also record a pointer to the rule in a data structure associated with the assertion. This step requires examination of each antecedent in each rule, and is therefore O(ar). At the end of the step, additional assertions have a zero alternative count (follows from lemma 2). The graph can be completely constructed and labelled in one step for each possible label, bounded by the maximum depth of reasoning. Construction of the labelled dependency graph is therefore O(ard).

Production of the decision table is done by repeated traversal of sections of the dependency graph. The cost of a single traversal is highly dependent on the details of data representation, but requires at most examination of each antecedent in each rule, therefore is at most O(ar). One traversal is required for each row in the resulting decision table. It is therefore necessary to estimate the number of rows.

There will certainly be one row in the decision table for each rule whose consequent is a conclusion. Additional rows will arise from alternative paths for satisfying the antecedents of one of these *terminal* rules. An alternative arises if one of its antecedents is the consequent of more than one rule. It follows that the number of rows is equal to the number of terminal rules if the *alternatives count* is *1* for each assertion, and that the number of rows increases in a complex multiplicative way as the *alternatives counts* become greater than *1*.

The problem is the same as converting an arbitrary boolean expression into disjunctive normal form. For example

(a + b & (c + d)) & (e + f)

converts to

a & e + b & c & e + b & d & e +
a & f + b & c & f + b & d & f

We can compute the number of rows during the construction and labelling of the dependency graph. We need an additional data structure *full alternative counts* paralleling the *alternative counts*, having one entry for each assertion, and also for each conclusion. For a particular assertion, the latter is the number of rules having that assertion as consequent, while the former will be the number of disjuncts in the disjunctive normal form expression which implies that assertion starting from facts only. *Full alternative count* is a sort of transitive closure of*alternative count*, and is initially *0*. A fact has *full alternative count* of *1*.

When a rule is labelled, the *full alternative count* associated with its consequent is increased by the product of the full alternative counts of its antecedents. The total number of rows in the decision table is the total of the *full alternative counts* of all the conclusions.

If *N* is the number of rows, then the production of the decision table at most O(Nar). This step will tend to dominate the computation time.

Backward Chaining

The basic algorithm given above is based on a forward chaining inference engine, which starts with known facts and derives conclusions as the antecedents of rules become known. If the rules labelled *n* are identified as *layer n*, This corresponds to a traversal of the dependency graph starting from layer *1*. This traversal will tend to be layer by layer, necessarily so if negation by failure is employed. We will call the labels and layers obtained in the forward chaining approach as *forward* labels and layers, respectively.

An alternative way to inference is to start from the rules with conclusions as consequents and work backwards, called *backward chaining*. If the inference engine is backward chaining, we note that a graph can be verified acyclic by successively removing nodes with no output arc. The dependency graph can therefore be labelled with minimum distance from K rather than from maximum distance from B, and the algorithm modified accordingly. The maximum depth of reasoning d is clearly unchanged. These labels will be called *backward* labels.

Note that if a node has backward label i it must have forward label less than d - i. This follows from the observation that step d of the backward algorithm removes nodes with no input arc. The backward collapse therefore has the same result as the forward collapse since it can be performed by the forward collapse algorithm on the graph with the backward labelling.

Under the assumptions made so far, forward and backward chaining have exactly the same result, obtained in the forward chaining case by collapsing the dependency graph from the facts onto the conclusions through the assertions. In the backward chaining case, the assertions are subgoals, and the algorithm collapses the graph from the conclusions onto the facts through the subgoals. The two strategies can be viewed as alternative ways of constructing the function mapping the set of assignments into the set of subsets of conclusions.

Negated Assertions in Antecedents

We assume that the inference engine does not evaluate a rule with a negated assertion in its antecedent until it has evaluated all rules with that assertion as consequent. Since no rule has a negated assertion as a consequent, the inferencing must rely on the closed world assumption for negation.

Let r be such a rule and let i be the maximum forward label of any negated assertion in its antecedent. Rule r is labelled with the maximum of i and the label of any un-negated assertion in its antecedent. When rule r is reached in the forward collapse algorithm, any negated assertion is replaced by the negation of the expression implying that assertion.

Negated Assertions as Consequents

In a system where negated assertions are allowed as consequents, the closed world assumption is not needed. An assertion and its negation are in most respects separate propositions and can be treated as such, with two exceptions. First, the negation of an assertion can not label any arc leading from the base set to a rule for which that assertion is a consequent. Second, any input which would imply both the assertion and its negation is forbidden. The algorithm in its course identifies an expression in facts which implies each proposition. If we have for assertion a

E_1 -> a; E_2 -> not a;

then a valid input must be consistent with

not(E_1 and E_2)

Test Case

The algorithm has been successfully applied to the Garvan ES1 thyroid assay system (Buchanan 1986), which has been in routine use since 1984. It has 600 rules, and an average depth of reasoning of about 4. Some of the rules have negated assertions in their premises, but no rule asserts a negation. The system normally runs on a PDP-11 with a specialized inference engine. For purposes of comparison, it was translated into OPS-5 and run on a Microvax II, where it operates at about 18 rule firings per second, taking about 220 milliseconds to generate a conclusion. It was transformed into a decision table with 5300 rows. There are 34 variables with a total of 93 possible values, so the decision table requires 5300 x 93 bits, or about 62k bytes of memory.

There are a number of ways to process a decision table. One is to convert it into a decision tree, using methods like ID3 (Quinlan 1982). This approach is presently under investigation. A balanced decision tree with N leaves identifies a particular leaf in $\log_2(N)$ decisions, so that a table with 4096 rows would be computed in 12 decisions, each of which is a simple *if...then* statement. This approach would clearly be extremely fast on standard hardware. In addition, there is evidence that the decision table can be considerably reduced, also the subject of continuing research.

Execution results presented here are from a method using an inexpensive bit-serial content-addressable memory (Colomb & Allen 1989) acting as a co-processor on a Sun 3/160. It is capable of processing a decision table at a rate of about 100 million bits per second, and can compute a decision from the transformed Garvan ES1 in about 2 milliseconds. The processor used has a programming model similar to the *MasPar*, *Distributed Array Processor*, and the *Connection Machine*, all of which are commercially available fine-grained parallel machines. The *MasPar*, for example, would be able to execute the system in about 20 microseconds.

We can conclude from this that it is possible to transform a general propositional expert system into a form that is capable of execution in a time sufficiently short that it opens many possibilities for the use of expert systems in real time applications.

Generalizations

There are a number of practical issues in expert systems engineering which are not addressed by the preceding results. These include explanation capability, obtaining expensive facts only when necessary, and inexact reasoning. The results can be generalized to deal with all of these issues.

Explanation Capability

An important feature of expert systems is the ability to explain a conclusion or the reasons for asking a particular question. Most approaches to explanation follow the chain of assertions between the base facts and the conclusion, so are derived from the trace of the traversal of the dependency graph.

In practice, many expert systems do not use explanations in their normal execution. (Garvan ES1, for example, is a batch program.) Jansen & Compton (1988) make a strong case for the separation of the normal execution environment where explanations are not available from a maintenance environment where a very complete explanation environment is provided.

In any case, it is possible to adapt the main results to give an efficient computation structure which permits a complete explanation capability. Rather than a decision table, this approach relies on executing the rules in sequence in a single pass.

Recall that the algorithm labels each rule with the maximum number of inference steps between it and the base facts. From lemma 1, all antecedents of rules at level i are determined by rules of level less than i. In addition, rules at the same level can be evaluated in any order. Clearly, if the rules are sorted by level, it is possible to execute them in a single pass. It is only necessary to keep a table of the value of each of the assertions (initialized to *false* if the closed world assumption is used) which is updated by any rule firing whose consequent makes that assertion. Since no assertion found in the antecedent of a rule can be changed by any subsequent rule, a complete explanation capability is available. For example, if the question is "why did a particular rule not fire?", the table of assertions will contain the values of all the antecedents of that rule at the time it was considered. A further explanation can be obtained in a similar way be examining the rules in earlier layers which have a particular assertion as consequent.

One way to represent this system is as a decision table with one row per rule and one column for each possible value of each fact augmented by one column for each possible value for each assertion. The Garvan system noted above can be represented as a decision table with 600 rows. There are 52 assertions, so 104 columns are needed besides the 93 required for the 34 fact variables, so that 600 x (104 + 93) bits or about 15k bytes are needed for its storage, considerably less than the 62k bytes needed for the fully expanded decision table.

The Knowledge Dictionary of Jansen & Compton (1988) has been re-implemented with an inference engine employing the method of this section (Lee, 1990). Note that the Garvan ES1 inference engine (Horn *et al.* 1985) takes essentially this approach to get fast execution on a PDP-11 with limited memory. Their approach can now be seen to be quite general.

Expensive Facts

The previous results make the implicit assumption that all facts are available at the beginning of inference, and all facts have equal cost. In practice, some facts may have a high cost, perhaps because they require database access or questioning the user. In this case, it is usual to first make use of the inexpensive facts available at the beginning, obtaining the expensive facts only if necessary. There will usually be rules whose consequent is not an assertion, but a command to assign values to a group of variables.

The set of facts can be labelled in the same way as the assertions, with facts available immediately labelled *zero*. In the decision table representation, the row headings can be sorted in increasing order of label. If the table is processed left to right, by the time a column labelled *one* or more is reached, the conditions under which that column is needed would be able to be evaluated. In the single-pass representation, the rule whose consequent is to obtain these facts will be in its correct place in the sequence.

Note that in this case the choice of forward or backward chaining affects the sequence in which expensive facts are obtained, since the dependency graph is traversed in a different order. This order is preserved in the sequence of column headings in the resulting decision table or in the sequence of rules in the single pass version.

Inexact Reasoning

Some expert systems use one or another form of inexact reasoning. The result can be adapted to this situation, although insufficient research has been conducted to determine the practicality of the method.

First, an uncertainty measure can be appended to each proposition. An assignment of values to variables would also assign an uncertainty measure. The subset of conclusions would also have uncertainty measures. The main theorem still holds.

Second, in the forward chaining algorithm, the uncertainty measure can be propagated as a tree of function composition. For example, if u(x) is the uncertainty of proposition x, we might have

a & b -> c	u(c) = f(u(a), u(b))
c & d -> e	u(e) = f(u(c), u(d))

then we would have

$$u(e) = f(f(u(a), u(b)), u(d))$$

If the uncertainty propagation function is associative, it is not necessary to record the tree of inferences by which the assertions are eliminated, and the uncertainty of a conclusion can be computed directly from the uncertainties of the base facts in its antecedent.

In particular, the commonly employed Bayesian measure of uncertainty is *a priori* independent of the intermediate assertions, since the joint probability of conclusions and base facts is known in principle independently of the reasoning system.

Advantages of Decision Table Representation

Representation of an expert system as a decision table has advantages apart from the possibility of faster execution.

Real Time

The main impact of these results on real-time systems is that execution time is not only much faster than conventional implementations, but it is also bounded. If the decision table is converted into a decision tree, the maximum number of decisions is known. If the decision table is processed directly by a fine-grained parallel processor, the maximum number of column operations needed is known.

A secondary benefit, particularly when the decision table is processed directly using a fine-grained parallel processor, comes from the fact that in real time situations facts are sometimes available asynchronously. In the decision table representation, it is very simple to compute the set of conclusions consistent with any assignment, no matter how incomplete. If we collect facts as they become available into what can be called a *current assignment*, we can always associate with the current assignment the set of conclusions consistent with it. Of course, we can in particular note the conclusions determined by the current assignment.

There might be a subset of conclusions considered to be important for some reason. It would be easy to monitor whether any important conclusions were consistent with the current assignment, for example. The possibility of one of these might trigger some additional measurements.

When a measurement is made which conflicts with a fact in the current assignment, the associated conclusions of the new current assignment can be computed quickly, perhaps making complex truth maintenance algorithms less necessary.

Rule Induction

Since propositional expert systems are equivalent to decision tables, it is more plausible that rule induction methods which build decision trees from examples (e.g Quinlan 1986), are generally applicable.

Consistency and Completeness

The decision table representation is much easier to test for consistency and completeness. The methods advocated by e.g. Cragun & Steudel (1987) are seen to be generally applicable.

Conclusion

The equivalence of propositional expert systems and decision tables has been shown, and a practicable algorithm presented for transforming an expert system into a decision table. The algorithm has been successfully tested on a substantial system of real utility. The method is capable of generalization to accommodate many of the practical problems encountered in practice, and makes consistency and completeness checking much easier. A particular consequence is that the computation time for these systems can be reduced by orders of magnitude, potentially greatly increasing the applicability of expert systems technology especially for real time problems.

Acknowledgements

Thanks to the referees for their helpful suggestions.

References

Beinat, P. & Smart, R. (1989) COLOSSUS: Expert Assessor of Third Party Claims *Fifth Australian Conference on Applications of Expert Systems* Sydney, Australia, pp. 70-85.

Buchanan, B. (1986) Expert Systems: Working Systems and Research Literature *Expert Systems* 3(1): 32-51.

Colomb, R.M. (1989) Representation of Propositional Expert Systems as Decision Tables *Third Joint Australian Artificial Intelligence Conference (AI'89)* Melbourne, Victoria, Australia.

Colomb, R.M. & Allen, M.W. (1989) Architecture of the Column Computer *Conference on Computing Systems and Information Technology*, Institution of Engineers, Australia.

Cragun, B.J. & Steudel, H.J. (1987) A Decision-Table-Based Processor for Checking Completeness and Consistence in Rule-Based Expert Systems *International Journal of Man-Machine Studies* 26(5):633-648.

Gupta, A., Forgy C., Newell, A, & Wedig, R. (1986) Parallel Algorithms and Architectures for Rule-Based Systems *13th Annual International Symposium on Computer Architecture* IEEE, 28-37.

Horn, K.A., Compton, P., Lazarus, L., & Quinlan, J.R. (1985) An Expert Computer System for the Interpretation of Thyroid Assays in a Clinical Laboratory *Australian Computer Journal* 17(1):7-11.

Jansen, R. & Compton, P. (1988) The Knowledge Dictionary: An Application of Software Engineering Techniques to the Design and Maintenance of Expert Systems *AAAI-88 Workshop on Integration of Knowledge Acquisition and Performance Systems*, Minnesota USA.

Lee, M. R.-Y. (1990) *The Implementation of a Knowledge Dictionary in SQL* Technical Report TR-FD-90-02 CSIRO Division of Information Technology, Sydney Australia.

Nguyen, T.A., Perkins, W.A., Laffey, T.J., & Pecora, D. (1985) Checking an Expert System Knowledge Base for Consistency and Completeness" *IJCAI-85* Morgan Kaufman.

Quinlan, J.R. (1982) Semi-Autonomous Acquisition of Pattern Based Knowledge, in Hayes, J.E., Michie, D., and Pao, Y-H, eds, *Machine Intelligence 10*, Ellis Horwood, 159-172.

Quinlan, J.R. (1986) Induction of Decision Trees *Machine Learning* 1(1): 81-106.

Stolfo, S.J. (1985) On the Design of Parallel Production System Machines: What's in a LIP? *Proceedings 18th Hawaii International Conference on System Science.*