ELSEVIER

Artificial Intelligence 109 (1999) 187–209

Artificial Intelligence

www.elsevier.com/locate/artint

Representation of propositional expert systems as partial functions

Robert M. Colomb [1]

Department of Computer Science and Electrical Engineering, The University of Queensland, Queensland 4072, Australia

Received 10 February 1998; received in revised form 6 December 1998

Abstract

Propositional expert systems classify cases, and can be built in several different forms, including production rules, decision tables and decision trees. These forms are inter-translatable, but the translations are much larger than the originals, often unmanageably large. In this paper a method of controlling the size problem is demonstrated, based on induced partial functional dependencies, which makes the translations practical in a principled way. The set of dependencies can also be used to filter cases to be classified, eliminating spurious cases, and cases for which the classification is likely to be of doubtful validity. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Propositional systems; Machine learning; Decision tables; Decision trees; Knowledge filtering

1. Introduction

There is a large class of expert systems whose purpose is essentially to classify cases, for example, to diagnose disease from symptoms. Several different methods are used to represent the knowledge in such systems, and to form the basis of computer implementations, including propositional production rules, decision tables and decision trees.

It has been known for some time that these representations are equivalent, in that a system represented in one can be automatically translated into another, exactly preserving the input–output behaviour of the original representation. This issue is addressed in detail by Colomb and Chung [5]. Translatability means that a body of knowledge can be represented in different ways for different purposes. For example, an expert system may be

[1] Email: colomb@csee.uq.edu.au.

PII: S0004-3702(99)00013-2

built by experts as a set of rules, which might be a natural way for humans to understand the knowledge; but a decision table representation can make it much easier to perform an analysis of the vulnerability of the knowledge base to measurement errors [3]; and a decision tree representation may give a fast bounded time implementation [19,20]. On the other hand, an expert system may be built originally as a decision tree, say by induction from a set of cases [15,16], but be translated into one of the other forms for analysis or understandability. Since these various representations of knowledge are interchangeable, we will call them all *decision objects*.

However, even though the translations can be automatically performed by simple algorithms, there is a severe practical difficulty, namely that the translated representation can be very much larger than the original. Furthermore, most of the constituent parts of the translated object may not be used in practice, even if all the parts of the original are.

The main result reported in this paper is to show why this problem of greatly increased size occurs and to show a simple method of representing decision objects which eliminates it. To do this, we first exhibit a unified representation of the various forms of decision objects and present in this representation the translatability results from the literature. We then show how the translated representation becomes inflated, and that this inflation is the result of representing inherently partial functions on the attribute space as total functions. To solve the problem, we show a simple way of representing and estimating the domain of the partial classification function represented by the decision object, and show that by use of the knowledge of the domain, we obtain practical translation among forms of decision objects with little inflation. The paper concludes with a discussion and implications of the results.

2. Forms of decision object

A classification system typically processes a case consisting of measurements of a number of variables by assigning a classification to the case. This section is a collection and systematization of results from the literature. The proofs are therefore given somewhat informally since the full detail is available elsewhere.

2.1. Propositions

We interpret a variable as a measuring instrument, used by a computer system to monitor a real-world process of some kind. A measurement by a specific variable is the assignment of a specific value to that variable, notionally by the real-world process. The *value set* belonging to a variable is a discrete set of names, usually describing qualitative properties. A value set must have at least two members. The prototypical case is a Boolean variable with values $\{true, false\}$, but other value sets are possible: for example, the variable *sex* has the value set $\{male, female\}$. If a variable refers to a continuous measurement, its value set frequently names the results of a series of relational tests on the measurement: for example, the variable *tsh* might have the values *high*, *borderline_high*, *normal*, *borderline_low*, *low*, *missing*. There is no limit in principle to the cardinality of a value set. This notion of variable with finite value set is widely applicable: in particular, it is the basis for most expert systems.

In an application, there are generally a number of variables. More formally, there is a set $X = \{x_i\}$, consisting of at least one variable. This set of variables is interpreted as a view of the real-world process. Each variable x_i has a value set V_i. A state of the real-world process as measured by the system is an assignment $x_i = v_j$ for v_j in V_i for all variables. This measurement of state is referred to as a *case*. The set X will be called the *variable set* associated with the system under consideration.

An *elementary proposition* consists of a variable and a value from the value set belonging to that variable, designated $x_i = v_{ij}$ for v_{ij} in V_i. We will designate by p_i the set of elementary propositions p_{ij} associated with variable x_i. The elementary propositions associated with the variables $\{x_i\}$ will be referred to as *case elementary propositions*. A case is represented as a proposition by a *choice* from $\{p_i\}$, that is exactly one member of each p_i. The ith choice will be designated c_i and a case c will be represented by the set $\{c_i\}$. Each c_i is a case elementary proposition associated with variable x_i. Classification is a function whose domain is the set of possible choices from $\{p_i\}$, and whose range is a set of elementary propositions whose value set is a (typically small) finite set $D = \{d_j\}$. The classification assigned to a case will be represented as a proposition of the form class $= d_j$ for d_j in D, where *class* is a reserved variable name. Elementary propositions associated with the variable *class* will be referred to as *classification elementary propositions*.

Example 1. Consider a system with two variables x_1 (*sex*) with value set $\{\mathit{female}, \mathit{male}\}$ and x_2 (*pregnant*) with value set $\{\mathit{true}, \mathit{false}\}$. The case elementary propositions associated with each variable are $p_1 = \{\mathit{sex} = \mathit{female}, \mathit{sex} = \mathit{male}\}$ and $p_2 = \{\mathit{pregnant} = \mathit{true}, \mathit{pregnant} = \mathit{false}\}$. There are four possible choices from $\{p_i\}$, two of which are $\{\mathit{sex} = \mathit{male}, \mathit{pregnant} = \mathit{false}\}$ and $\{\mathit{sex} = \mathit{male}, \mathit{pregnant} = \mathit{true}\}$. D is the set $\{\mathit{normal}, \mathit{plausible}, \mathit{extraordinary}\}$. Assume that the classification assigned to the two cases exhibited are, respectively, the classification elementary propositions $\mathit{class} = \mathit{normal}$ and $\mathit{class} = \mathit{extraordinary}$, while the classification assigned to the two other choices are both $\mathit{class} = \mathit{plausible}$.

Definition 2. An elementary proposition is *binary* if its value set has cardinality 2.

Proposition 3. *The negation of an elementary proposition is a conjunction of elementary propositions.*

Proof. An elementary proposition is a formula of the form $x_i = v_{ij}$. If it is not true that $x_i = v_{ij}$, then it must be true that $x_i = v$, where $v \in V_i \backslash v_{ij}$, so that

$$\sim(x_i = v_{ij}) \equiv \bigvee_{k \neq j} x_i = v_{ik},$$

where k ranges over the indexes of the members of the value set V_i. □

Corollary 4. *The negation of a binary elementary proposition is a binary elementary proposition.*

2.2. Rules

A rule system form of decision object is a collection of production rules. Production rules are constructed from elementary propositions, classifications, and a distinguished set of propositions $A = \{a\}$ designated *intermediate elementary propositions*. Intermediate elementary propositions are distinguished by having variables distinct from the variables which can occur in cases, and not including the reserved variable *class*. The antecedent of a rule is a conjunction of case or intermediate elementary propositions, or the negations of either, while the consequent is either an intermediate or classification elementary proposition. There is a single consequent in each production rule, so that a rule system is a collection of propositional Horn clauses. A production rule system will be called *well-formed* if its dependency graph [2] is acyclic. The *dependency graph* for a system of production rules has one node for each rule. An arc is defined with source $N1$ and target $N2$ if the consequent of the rule at $N1$ appears in the antecedent of the rule at $N2$. Note that this requirement is stronger than stratification as defined in the cited work. That work labels an arc "positive" or "negative" according to whether the consequent of $N1$ appears as a positive or, respectively, negative literal in the antecedent of $N2$. A stratified system has no cycles including a negative arc, but allows cycles all of whose arcs are positive.

The semantics of a propositional production rule based decision object is based on datalog under negation as failure. A set of propositional Horn clauses is trivially a datalog intensional database (IDB), since there are no variables. The extensional database (EDB) consists of a set of predicates each of which corresponds to one of the elementary case propositions. Each of the EDB predicates either contains one tuple, which consists of the token "true", or is empty. The standard stratified naive bottom-up evaluation procedure populates the IDB predicates by propagating the "true" token. The classifications assigned to a case are the classification elementary propositions which are not empty. All this is a straightforward application of the principles of datalog as described, for example, by Ullman [21,22].

Example 5. We will re-cast Example 1 as a datalog system. The IDB consists of the three Horn clauses:

$$(\mathit{class} = \mathit{normal}) \text{ :- } (\mathit{sex} = \mathit{male}), (\mathit{pregnant} = \mathit{false})$$
$$(\mathit{class} = \mathit{extraordinary}) \text{ :- } (\mathit{sex} = \mathit{male}), (\mathit{pregnant} = \mathit{true})$$
$$(\mathit{class} = \mathit{plausible}) \text{ :- } (\mathit{sex} = \mathit{female}),$$

and assume the EDB represents the case of a pregnant male:

$$(\mathit{sex} = \mathit{male}) = \{\mathit{true}\}$$
$$(\mathit{sex} = \mathit{female}) = \{\,\}$$
$$(\mathit{pregnant} = \mathit{true}) = \{\mathit{true}\}$$
$$(\mathit{pregnant} = \mathit{false}) = \{\,\}.$$

The datalog evaluation produces the following population of IDB predicates

$$(\mathit{class} = \mathit{normal}) = \{\,\}$$
$$(\mathit{class} = \mathit{extraordinary}) = \{\mathit{true}\}$$
$$(\mathit{class} = \mathit{plausible}) = \{\,\}.$$

The datalog semantics of rule systems shows that the well-formedness criterion which excludes recursion entirely involves no loss of generality.

Theorem 6. *For every stratified propositional IDB P there is an IDB with an acyclic dependency graph which has the same perfect model.*

Proof. Based on the datalog naive evaluation procedure. This procedure starts with the given EDB and no population for any IDB predicate, and proceeds from stratum to stratum (making sure that clauses containing negated predicates in their antecedents are not evaluated until all the clauses with those predicates as consequents are fully evaluated). Each step evaluates all clause bodies and possibly generates tuples for some of the IDB predicates. This evaluation is called applying the T operator. The initial population is named $T(0)$, and the population after n steps is called $T(n)$. It terminates when a fixed point is reached: $T(n+1) = T(n)$. $T(n)$ is the perfect model. □

Lemma 6.1. *In a propositional IDB, each predicate generates tuples only once.*

Proof. Each predicate in the IDB can have at most one tuple. Therefore once a clause puts a tuple in a predicate, it is fully evaluated. □

Even though a propositional predicate may be defined in several clauses, at most one clause firing fully evaluates the predicate.

Lemma 6.2. *For a given EDB, the evaluation procedure selects an acyclic subset of IDB clauses, which are the only clauses to generate tuples.*

Proof. A clause can generate tuples only if all of its positive antecedent literals have populations, and none of its negative antecedent literals can possibly have populations. At each step n of the evaluation, include the clauses which generate tuples in $T(n)$ in the required subset. Designate the clauses added at $T(n)$ by $C(n)$, and the union up to n of the $C(n)$ by $S(n)$. Lemma 6.1 insures that a clause can be in at most one $C(n)$. That the required subset at the fixed point is acyclic can be seen by induction. Clearly $S(1)$ is acyclic, since the only clauses which can generate tuples in $T(1)$ must have antecedents entirely in the EDB. If $S(n)$ is acyclic, then $C(n+1)$ includes only clauses all of whose antecedents contain only positive literals defined in $S(n)$ and negative literals which are empty and which have been fully evaluated, so $S(n+1)$ can have no cycles. □

Each EDB e generates an acyclic subset of the IDB by Lemma 6.2. Designate that subset by $S(e)$.

Lemma 6.3. *If e is an EDB and S' is an IDB which generates no tuples when evaluated with EDB e, there is an IDB with an acyclic dependency graph which has the same perfect model as S' when evaluated with EDB other than e and the same perfect model as $S(e)$ when evaluated with EDB e.*

Proof. By construction. If $S(e)$ generated no tuples when evaluated with EDB other than e, and $S(e)$ had no intermediate elementary propositions in common with S', then $S' \cup S(e)$ would be the desired IDB. In general, this is not the case. We need to isolate $S(e)$ from other EDBs and to make sure all intermediate elementary propositions in $S(e)$ do not occur in any other clause in the IDB.

It is easy to specify an additional set of rules which will serve to identify EDBs. A propositional EDB is a pattern of presence or absence of a token in a fixed collection of predicates. For EDB e, we can create a rule with consequent $i(e)$ which is true if the EDB is the pattern e and empty otherwise. The antecedent of *identification predicate* $i(e)$ has a positive literal for every proposition having a token in EDB e, and a negative literal for every proposition lacking a token.

Also, since all elementary propositions assigned values in $S(e)$ are assigned values by clauses in $S(e)$ together with EDB e, we can replace the names of the variables associated with all intermediate propositions in $S(e)$ with names tagged by e. We will call this process *isolation of an IDB.*

For $S(e)$, construct an IDB $S'(e)$ by including $i(e)$ as a conjunct in the antecedent of all clauses in $S(e)$. The only IDB which can generate tuples when evaluated with EDB e is $S'(e)$. Further, $S'(e)$ has the same perfect model as $S(e)$ (excluding the introduced identification predicates). If $S'(e)$ is also the isolation of $S(e)$, then $S' \cup S'(e)$ is IDB required for the lemma. □

The identification predicate is what really does the job. The only reason for worrying about isolation is that the dependency graph is defined in terms of the propositions which are the consequents of clauses, not clauses in isolation. Even though the introduction of the identification predicates is enough to keep $S'(e)$ from any influence in the evaluation of any other EDB, the isolation is required to separate the dependency graph.

Proof of Theorem 6. Construct the finite set (cardinality N) of all possible choices from $\{p_i\}$ as the set of possible cases C. Impose an ordering on C, yielding a sequence of cases with the kth member identified by c_k, $k \leqslant N$. For each c_k construct $S(c_k)$. Construct $S''(0)$ as the empty IDB. For each $k > 0$ construct $S''(k)$ by the application of Lemma 6.3 to $S''(k-1)$ and c_k. $S''(N)$ is the required IDB, which has an acyclic dependency graph and has the same perfect model as P for every possible case. □

Theorem 6 is of course not a practicable procedure to convert recursively defined propositional production rule sets to nonrecursive production rule sets. However, it shows that the use of recursion adds no expressive power. Therefore it is reasonable to suggest to the programmer of a recursively defined propositional production rule set that the set be re-implemented to avoid recursion.

Definition 7. Two decision objects are equivalent if they have the same perfect model for all EDBs.

2.3. Decision table

A decision table consists of a two-dimensional array of cells. Associated with each row in the array is a classification. The ith cell in a row is a nonempty disjunction of elementary propositions from the set of elementary propositions associated with the ith variable. A row in a decision table can be viewed as a rule whose antecedent is a conjunction of cells, and whose consequent is a classification. Therefore, a decision table can be viewed as a conjunction of row rules.

Example 8. The following is a decision table equivalent to the system in Example 5.

Row	Sex	Pregnant	Classification
1	$\mathit{sex} = \mathit{male}$	$\mathit{pregnant} = \mathit{true}$	$\mathit{class} = \mathit{extraordinary}$
2	$\mathit{sex} = \mathit{male}$	$\mathit{pregnant} = \mathit{false}$	$\mathit{class} = \mathit{normal}$
3	$\mathit{sex} = \mathit{female}$	$\mathit{pregnant} = \mathit{true} \vee$ $\mathit{pregnant} = \mathit{false}$	$\mathit{class} = \mathit{plausible}$

Definition 9. A cell is a *don't care cell* if it is the disjunction of all the elementary propositions in the set associated with its column.

The cell in Example 8 row 3 associated with the variable *pregnant* is a don't care cell.

Theorem 10. *An acyclic rule system is equivalent to a decision table.*

Proof. (Detailed in [5].) Essentially done by successively unfolding (partially evaluating) the production rules, replacing the intermediate propositions by formulas consisting entirely of case elementary propositions or their negations. The result is a conjunction of rules whose consequents are classification elementary propositions. The result is expressed in clausal form. These clauses are rows in a decision table, with the exception that a given row may contain no elementary proposition from the set associated with a given variable. If this is the case, the antecedent of the clause is augmented by a conjunct consisting of the don't care cell associated with the variable, which evaluates to *true*. □

Definition 11. A decision table is *unambiguous* if no assignment of truth values to elementary propositions can assign *true* to more than one classification elementary proposition.

An ambiguous decision table can arise from several sources. Decision tables arising from Theorem 10 can be ambiguous due to errors in the rule set. One motivation for using Theorem 10 is to check the set of rules for ambiguity, which is very difficult to see in the rule representation. It can also be ambiguous only for impossible cases. Insulating a system from impossible cases is one of the primary motivations of this paper. Sometimes decision tables are used to identify situations in which there is normally one classification, but sometimes can have more-multiple disease states for instance. In this situation, we can without loss of generality replace the variables in the classification elementary propositions with Cartesian products of variables, and the values by Cartesian products of values, so that the form of the elementary proposition is preserved and the table becomes unambiguous, preserving the remainder of the present theory.

Definition 12. A decision table is complete if every assignment of truth values to elementary propositions assigns *true* to at least one classification elementary proposition.

Observation 13. A rule system can be equivalent to an ambiguous and/or incomplete decision table. See [5].

2.4. Decision tree

A decision tree is a tree consisting of nodes of two classes: *deciding nodes* and *leaf nodes*. Each deciding node is associated with a variable and each leaf node is associated with a classification elementary proposition. Each arc has as its source a deciding node, and is associated with a case elementary proposition from the set associated with its source (*associated arc proposition*). Our formulation is somewhat unusual in that each leaf node is associated not only with a classification elementary proposition but with a proposition composed from case elementary propositions. The operational semantics is that a leaf node fires on a case if all the arc propositions in a path to the root are consistent with the case, and so is the leaf proposition. A decision tree may therefore fail to classify a case.

Example 14. The following is a decision tree equivalent to the table in Example 8.

Node 1 *sex—female—Leaf* 2 $(\mathit{pregnant} = \mathit{true}) \vee (\mathit{pregnant} = \mathit{false}) \rightarrow (\mathit{class} = \mathit{plausible})$
| *male*
Node 3 *pregnant—true—Leaf* 4 $\mathit{true} \rightarrow (\mathit{class} = \mathit{extraordinary})$
| *false*
Leaf 5 $\mathit{true} \rightarrow (\mathit{class} = \mathit{normal})$.

Definition 15. A decision tree is *complete* if each deciding node is the source of arcs associated with all of the elementary propositions associated with its variable, and all of its leaf propositions are equivalent to *true*.

Definition 16. A *path proposition* is associated with a path from the root to a leaf, and is the conjunction of all the associated arc propositions conjoined with the leaf proposition.

Proposition 17. *A conjunction of elementary propositions is either inconsistent or a choice from the sets of elementary propositions associated with a subset of the variables.*

Proof. Two distinct elementary propositions associated with the same variable are inconsistent. Therefore, any consistent conjunction contains at most one elementary proposition associated with each variable. □

Proposition 18. *A path proposition is equivalent to a decision table.*

Proof. The leaf proposition can be expressed in disjunctive normal form. The path proposition is therefore the conjunction of each of the leaf proposition disjuncts with all the arc propositions. By Proposition 17, each of the resulting disjuncts contains at most one elementary proposition associated with each of the variables associated with the deciding nodes in the path and the variables in the path propositions, or is inconsistent. If inconsistent, the disjunct can be removed. Finally, each disjunct can be augmented with don't care cell propositions associated with any variable lacking any associated elementary proposition in the disjunct. □

Theorem 19. *A decision tree is equivalent to an unambiguous decision table, which is complete if the tree is.*

Proof. Each distinct path through the decision tree is equivalent to a decision table each row of which is associated with the path's leaf classification elementary proposition. Each path proposition is inconsistent with every other path proposition, since every pair of paths must share at least one deciding node and the arcs in each path whose source is that deciding node have different and therefore, inconsistent associated arc propositions (which are are case elementary propositions with the same variable and a different value for each arc). The union of the decision tables from all paths is a decision table, which is unambiguous, since each leaf and therefore, path has a single classification.

If the decision tree is complete, then each deciding node partitions the set of possible cases, so there is a partition of the set of possible cases where each partition is associated with a leaf. The leaf proposition is *true*, so that every case in that partition is consistent with the decision table equivalent to the path proposition associated with the leaf. Each case in the population of possible cases is therefore, contained in one of the parts of the partition, and is therefore, consistent with at least one row in the decision table. □

Theorem 20. *An unambiguous decision table is equivalent to a decision tree, which is complete if the table is.*

Proof. By induction on the number of attributes associated with the table. We adopt the convention that a decision table with no attributes has a single row with the propositional value *true*, and therefore a single classification.

Basis step: If any of the following is the case, then the induction terminates with the indicated action:

1. The number of attributes is zero. Create a leaf node whose classification is the table's single classification elementary proposition, and whose leaf proposition is *true*. (Depends on the table being unambiguous.)
2. The number of rows is zero. Create a leaf node whose classification elementary proposition is arbitrary and whose leaf proposition is *false*.
3. The number of distinct classifications is one. Create a leaf node whose classification elementary proposition is the table's unique classification, and whose leaf proposition is the disjunction of propositions created from each row by the conjunction of the cell propositions.

Induction step: There is at least one attribute, at least one row, and at least two distinct classifications in the table T.

(1) Select an attribute a by some method (this point is discussed below).
(2) Create a deciding node associated with this attribute.
(3) For each v in the value set of a, create:
- An arc whose source is the deciding node created and whose arc proposition is $a = v$.
- A decision table $T(a, v)$ derived by removing the cell associated with a from each row of T in which the cell associated with a is consistent with $a = v$.

$T(a, v)$ has one fewer attribute in its associated attribute set than T, so the induction proceeds.

The tree is not complete unless every deciding node is the source of an arc associated with each elementary proposition associated with its variable. The arcs are created by part (3) of the induction step. If one of the elementary propositions is missing, then the table has no row consistent with that elementary proposition and no case with that elementary proposition as a conjunct will be classified. Similarly, if the leaf proposition does not evaluate to *true*, then no case consistent with the path proposition conjoined with the negation of the leaf proposition will be classified by the table, so the table must be incomplete. The tree is therefore, complete if the table is. □

From a practical point of view, the key to the algorithm developed from this theorem is in the method of choosing an attribute in part (1) of the induction step. Shwayder [19,20] uses a heuristic based on equalizing the number of rows in the $T(a, v)$ (*maximum dispersion*), giving one of the standard algorithms for translating from a decision table to a decision tree. Quinlan [16] uses a heuristic based on minimizing the maximum number of distinct classifications in the $T(a, v)$ (*maximum entropy gain*). Theorem 20 with the maximum entropy gain selection procedure gives a variant of the ID3 algorithm.

Theorems 19 and 20 show that decision trees and unambiguous decision tables are exactly equivalent.

2.5. Cases

Quinlan's 1986 paper [16] is based on the concept of a training set of cases, and is intended to construct a decision tree from a training set.

Definition 21. A *training set* is a set of cases. Associated with each case is a classification.

Proposition 22. *A training set is a decision table.*

Proof. By inspection. □

Definition 23. A training set is *noise-free* if it is unambiguous.

Observation 24. Theorem 14 applied to a noise-free training set using the maximum entropy gain selection procedure is the ID3 algorithm.

2.6. Ripple-down rule trees

Another form of propositional knowledge representation is the ripple-down rule tree [7]. A ripple-down rule tree is a binary tree consisting of a set of *decision nodes* and a set of *leaf nodes*. A decision node is associated with a conjunction of elementary propositions. A leaf node is associated with a classification elementary proposition. There are two arcs whose source is a decision node. The *true* arc is taken if the node proposition is consistent with a case, the *false* arc otherwise.

Example 25. The following is an example of a ripple-down rule tree equivalent to the decision tree in Example 14:

$$\begin{array}{l}
\mathit{Node}\ 1\ \mathit{if}\,(\mathit{sex}=\mathit{male})\,\&\,(\mathit{pregnant}=\mathit{true})\text{—}\mathit{true}\text{—}\mathit{Leaf}\ 2\ (\mathit{class}=\mathit{extraordinary})\\
\qquad\qquad\qquad\ \ |\,\mathit{false}\\
\qquad\quad \mathit{Node}\ 3\ \mathit{if}\,(\mathit{sex}=\mathit{female})\text{—}\mathit{true}\text{—}\mathit{Leaf}\ 4\ \ (\mathit{class}=\mathit{plausible})\\
\qquad\qquad\qquad\qquad\quad\ |\,\mathit{false}\\
\qquad\qquad\qquad\qquad\quad\ \mathit{Leaf}\ 5\ (\mathit{class}=\mathit{normal}).
\end{array}$$

Definition 26. An *rdr path proposition* associated with a path through a ripple-down rule tree is constructed from a path from the root to a leaf. If the path takes a *true* arc, then the proposition associated with the decision node is a conjunct. If the path takes a *false* arc, then the negation of one of the conjuncts in the associated proposition is a conjunct.

Proposition 27. *Without loss of generality, an rdr path proposition is a decision table row.*

Proof. An rdr path is a conjunction of disjuncts. Each disjunct is a disjunction of elementary propositions associated with the same variable, since disjunctions are either single elementary propositions or are negations of elementary propositions (Proposition 3). By Boolean algebra, the rdr path can be represented as a disjunction of conjunctions of elementary propositions. By Proposition 17, each disjunct is either a choice from a set of variables or inconsistent. An rdr path is therefore, either inconsistent or a conjunction of disjuncts of elementary propositions associated with distinct variables. If the rdr path lacks elementary propositions associated with a given variable, the corresponding don’t care proposition can be conjoined. The classification is the classification associated with the leaf node. □

Theorem 28. *A ripple-down rule tree is equivalent to an unambiguous decision table.*

Proof. The decision table is the union of all rdr path propositions. Note that there can be many rdr path propositions associated with a single leaf, if the path from the root to the leaf takes any *false* paths whose source decision node is a nontrivial conjunction of elementary propositions, so there may be many rows in the decision table with the classification associated with a given leaf. However, rows with classifications from distinct leaves are inconsistent, since there must be a decision node in common, with one path taking the *true* arc and one the *false* arc. The table is therefore unambiguous. □

Section 2 has presented a framework in which the equivalences among the various forms of decision object (rules, decision tables, decision trees, cases and ripple-down rule trees) can be easily seen. Using these equivalences, one can develop a classification system using a form of decision object convenient for the developers, then translate it into other forms for purposes of analysis and execution.

3. What is wrong with this picture?

The idyllic theory described in the previous section encounters problems when put into practice. Section 3 describes the problems encountered and analyses their cause.

Some terminology is needed. We will say that a decision object has a number of parts. What a part is depends on the form of decision object. For a set of rules, a part is a rule; for a decision table, a row; for a decision tree or ripple-down rule tree, a leaf. The *size* of an object is the number of parts it has.

A decision object is designed to solve a practical classification problem. We would expect that when we build a decision object, no matter what its size, that all of its parts will be used in practice. A part that is not used represents a cost giving no benefit. We will say that a decision object is *compact* to the extent that all of its parts are used in practice (measured, for example, by monitoring the decision object over a long period of use). Generally speaking a decision object in its constructed form will be compact.

The problem we encounter is that when an object is transformed, the new object can be very much larger than the original, and much less compact. We designate this phenomenon as *inflation*.

For example, the transformations from Theorems 10, 20 and 28 were applied to a well-known expert system Garvan ES1 [11], which constructed clinical interpretations of thyroid hormone assays in a hospital pathology laboratory. It was in clinical use from about 1984–1990, and was applied to many thousands of cases per year. The compactness of the various forms was tested using a sample of 9805 cases, representing more than one year's use towards the end of the system's life when its accuracy was more than 99%.

Garvan ES1 was constructed as a set of production rules, of which at the end of its life there were 661, all of which were exercised by the set of cases. The system was, as one might expect, very compact. Applying Theorem 10, the production rules were transformed into an equivalent decision table which had 5286 rows. Only 653 of these rows were fired by any of the year's cases. The transformed object is much larger than the original, and much less compact.

Theorem 20 was then applied to the 653-row subset of the transformed decision table consisting of the rows which were fired by any of the year's cases, using the maximum dispersion selection procedure, resulting in a decision tree with 30,693 leaf nodes, of which only 807 were fired by any of the 9805 cases. The inflation is much worse than in the previous situation.

Finally, Compton and Jansen [7] produced a ripple-down rule version of the knowledge base (called Garvan RDR) which had 557 leaves in its ripple-down rule tree, all of which were necessarily exercised in practice since the tree was constructed in a process based on the sample of cases. Application of the algorithm derived from Theorem 28 would result in a decision table with 481,803,735 rows. One would expect, of course, that some of these rows would be inconsistent or would be subsumed by other rows. Nevertheless, only 680 of the rows fire on any of the cases, giving an extreme inflation.

Inspection of the proofs of the theorems shows how the transformed object gets bigger than the original. Theorem 10 translates a set of production rules to a decision table by progressively substituting the antecedents of intermediate propositions for their occurrences in other rules. If the antecedent is a disjunction, or appears negated in a rule, or both, a rule absorbing the intermediate proposition gets broken into several rules whose antecedents are conjunctions of elementary propositions. If the absorbing rule has itself an intermediate proposition as a consequent, the rules multiply. The algorithm for transforming rules to decision tables derived from Theorem 10 therefore produces a result whose size is exponential in the length of a chain of intermediate propositions. (Garvan ES1 had an average of three intermediate propositions between the measurements and the classifications.)

Translating from a decision table to a decision tree, part (3) of the induction step of Theorem 20 duplicates rows of the decision table where the cell associated with the selected variable contains a disjunction of elementary propositions. The algorithm derived from the theorem therefore, produces a result whose size is exponential in the number of variables. (Garvan ES1 had 34 variables, and the decision tree had an average path length from the root to a leaf of about 14.)

Finally, the number of rows in the decision table equivalent of a ripple-down rule tree is more than the product of the number of conjuncts in the propositions associated with the decision nodes in the longest chain of *false* arcs.

So we can see how inflation happens, but it remains to see why it happens. After all, the original decision object is created compact. We can see how it gets large, but where do all the unused parts come from?

To see this, we step back to the black box view of a classification expert system. The system can be seen as monitoring some real-world process which generates cases. The system observes the cases through its variables. Each distinct case is represented by a choice of values for the variables. The variables can therefore be seen to determine a space defined by the distinct possible choices. We call this space the *attribute space* for the process. It depends solely on the variables and their value sets.

The attribute space for a process can be quite large. A convenient measure of its size is the number of bits necessary to represent a case, disregarding the statistics of the process. If all the variables are Boolean, the size of the space is the number of variables. Otherwise, the size of the space is the total of the log to the base 2 of the cardinalities of the value sets. For example, Garvan ES1 has 34 variables, whose value sets range in cardinality from 2 to 6. Its size is 46 bits.

If Garvan ES1 has an attribute space of 46 bits, there are $2^{46} \approx 10^{14}$ possible distinct cases. At 10^5 cases per year and if all cases are different, it would take 10^9 years before all the cases were encountered. In practice, as one might expect, there are fewer than 4000 distinct cases in the year’s history, some of which occur more than 100 times. Where the attribute space is large, one would expect that the process has many variables which are contingent on others taking specific values (pregnancy is a relevant variable only if the sex of the patient is female, for example). Also, values of some variables occur in patterns depending on the values of others (the hormone profile of a pregnant woman or a child is different from a male adult, for example, and there are different characteristic disease states). In general, where the attribute space is large one would expect that almost every combination of values would produce a case the domain experts would regard as absurd. Even if there were 10^6 possible valid cases for the Garvan ES1 process, one would have to generate 10^8 cases at random before the expert would find one that makes sense.

This observation suggests that in systems with large attribute spaces the real-world process is confined to a very small region. The knowledge of the experts is confined to this small region, which we might call the *region of experience*. The expert system classifies cases, so can be seen as a function taking the region of experience to a space of classifications. This accounts for the compactness of the decision objects constructed.

These decision objects are, however, constructed as total functions on the attribute space, not as partial functions. The strategy works essentially because by definition the process being monitored never generates an impossible case. However, this strategy leaves the expert system vulnerable to errors or maliciousness. The famous experiment in which Lenat answered Mycin's questions as for a dead person, and Mycin happily diagnosed a specific course of treatment, is almost certainly a case where the input lay outside Mycin's region of experience.

Decision objects get their compactness essentially by liberal use of don't care propositions, or by conjunctions of elementary propositions which fail in limited ways. These elements are found and expanded by the transformation algorithms. Inflation is a side effect of the practice of constructing classification systems as total functions.

4. A cure for inflation

If inflation is a side effect of the construction of classification systems as total functions on large attribute spaces, then it would seem reasonable to look for a cure by defining the systems as partial functions. If one could get a characterisation K of the region of experience associated with a process generating cases to be classified, then we could use K to prune the transformed decision objects as they are being created. In Theorem 10 (rule → table), we would keep only those disjuncts of the definition of an intermediate proposition which are consistent with K. In Theorem 20 (table → tree), part (3) of the induction step would only make copies of rows which are consistent with K. In Theorem 28 (rdr tree → table), we would keep only those rdr paths which are consistent with K. A good estimate of K would not only control inflation, but would act as a filter detecting cases which are either spurious or perhaps legitimate but outside the experience of the domain experts.

A natural way to represent K is by a set of constraints stating that the values of certain variables are determined by the values of others. These sort of constraints are partial functional dependencies (*PFDs*). The variables are in general independent, but if the set of variables in the domain of the dependency take on a particular combination of values, then the value of the variable in the range of the dependency is fixed. For example, in general the variables *sex* and *pregnant* are independent. However, if *sex* takes the value *male*, then *pregnant* takes the value *false*. Similarly, if *pregnant* takes the value *true*, then *sex* takes the value *female* (at least in the experience of Garvan ES1).

Partial functional dependencies could be constructed by the domain experts as part of the process of building a decision object. Certainly, the *sex*/*pregnant* situation in the previous paragraph is pretty straightforward. However, many expert systems are constructed by induction using various methods to produce various forms of decision object. If a body of cases is available, then it is plausible to induce a set of partial functional dependencies.

In fact, one can see *a priori* that it must be possible to induce a set of partial functional dependencies. If the set of cases available is much smaller than the attribute space, then by assumption there are very many combinations of variable values which do not occur. The problem is therefore, not whether we can induce a set of PFDs, but whether we can induce a set of practical size which gives a sufficiently tight upper bound on the region of experience, whether the induction can be done at an acceptable cost, and whether the set induced is statistically reliable.

There are a number of algorithms available for estimating sets of PFDs. One is from the method of rough sets [13] which looks for rules resulting from value reducts, taking each attribute in turn as a decision attribute (set of classifications). Another comes from the data mining literature [1]. In data mining terminology, a PFD is an association with 100% confidence (no counterexamples).

We want PFDs with a small number of elementary propositions in their antecedents. There are two reasons for this. The first is that a single PFD with m elementary propositions in the antecedent and consequent taken together constrains the attribute space more the smaller m is. If all the variables are Boolean and there are n variables, then a single constraint reduces the attribute space by a factor of 2^{n-m+1}. The second reason for wanting PFDs with few propositions in their antecedent is that the algorithms for computing PFDs are exponential in the number of propositions in the antecedent. Small PFDs are therefore, both stronger and practical to compute.

Finally, we want PFDs which are statistically reliable. In the absence of a good theory of the statistics of PFDs, it seems reasonable to look for PFDs which have a large number of positive examples (in data mining terminology, associations with high support). This prevents rare cases from contributing possibly spurious PFDs.

The existence of a good set of PFDs is a question which can only be settled experimentally in particular situations. To gain experience, we estimated PFDs for Garvan ES1 from the set of 9805 cases (of which 3856 are distinct), using the rough sets library. [2] We applied the algorithm to the set of 3856 distinct cases, so that duplicated cases did not contribute to support. This process produced 382 PFDs with a single elementary proposition in their antecedent at a support level of 1 and 332 such PFDs with a support level of 10. Further, there were 13384 PFDs with two elementary propositions in their antecedent at a support level of 1 and 5858 such PFDs at a support level of 10. Since we are here concerned with the theoretical basis and computational feasibility of working with sets of PFDs, but want statistically reliable constraint sets, we chose the upper support level (10 examples with no counterexamples). We will call these two sets of constraints Garvan $K1$

[2] Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland.

and Garvan $K2$, respectively, and in general a set of constraints with one antecedent $K1$, and with two $K2$.

Further experience might suggest a different support level, but at least in this case the number of rules computed does not vary greatly with differences in support level so it would make little difference in the following.

This shows that it is possible, in at least one application, to induce a reasonable set of small PFDs. The remaining question is how strong they are—the size of the region of experience defined by them, and what difference they make to the inflation problem. This requires some more theory.

Definition 29. A proposition *covers* a case if the case is consistent with the proposition.

Definition 30. A variable is *missing* from a proposition if the proposition contains no elementary proposition associated with it.

One way to compute the strength is to use a method of Cragun and Stuedel [8]. In our terminology, they show that if a proposition is in disjoint disjunctive normal form (each disjunct is inconsistent with every other) the number of cases covered by the proposition is the sum of the number of cases covered by each disjunct. The number of cases covered by each disjunct is one if the disjunct is the conjunction of propositions associated with all variables, and the product of the cardinality of the value sets of the missing attributes otherwise.

Unfortunately, this method is of limited utility, since K is constructed in conjunctive normal form, and the process of conversion from conjunctive to disjunctive normal forms is exponential in the number of conjuncts. Garvan $K1$ requires the computation of $2^{332} \approx 10^{100}$ conjuncts, while Garvan $K2$ requires $3^{5858} \approx 10^{2610}$ conjuncts, clearly beyond the bound of computational feasibility, even for Garvan $K1$ only.

One might think that there might be considerable redundancy in K, in that some of the constraints might be deduced from others. This is particularly easy to test in the case of $K1$, since the redundancy can be viewed as a case of transitive closure. If $a \to b$ and $b \to c$ are both in $K1$, then the algorithms used will ensure that the redundant $a \to c$ is also. Applying this method to the 332-conjunct Garvan $K1$ reduces it to 296 conjuncts, taking the cost of conversion to disjunctive normal form from 10^{100} to 10^{89}, still well beyond computational feasibility.

Although there may be problems where the Cragun and Steudel approach is feasible, it is clear that there are some in which it is not. A Monte Carlo method based on constraint propagation is more generally applicable.

The basis of the Monte Carlo method is the generation of random cases. A naive approach would generate a population of random cases and count the cases which satisfy K. This is not generally practical, since we are expecting the region of experience to be a tiny fraction of the entire attribute space—in the case of Garvan ES1, the region of experience might be as small as 10^{-10} of the attribute space.

If we augment the generation of random cases with propagation of constraints, we obtain a feasible method of generating an estimate of the size of the region of experience.

Algorithm 31. Estimate the strength of a set of constraints K.

The algorithm is based on generating cases by successively choosing random values for variables and propagating these values, until all variables are assigned values. The average number of random bits needed to generate values is an estimate of the size of the space covered by K. A variable which is not assigned a definite value will be referred to as an *open variable*. Note that the method may restrict the possible values a variable may be assigned, but the variable remains open until a definite value is assigned (this last applies only to variables the cardinality of whose value set is greater than 2).

(1) When sufficient cases have been generated terminate, returning the average number of bits needed to generate a case.

(2) Generate a single case:
  - If all variables have values assigned, terminate, returning the number of bits needed to select values. Otherwise:
  - Select an open variable at random. Generate sufficient random bits to give this variable a definite value, and assign that value. (Note that the cardinality of the value set may be reduced due to earlier constraint propagation.)
  - Propagate the new value to the other open variables using K.

Algorithm 31 is a special case of the more general problem of propagation of finite domain constraints. It would be possible to build a constraint size estimating procedure using a finite domain constraint logic programming language [12].

Using an implementation of Algorithm 31 specifically designed for $K1$ constraint sets, Garvan $K1$ was found to be of size 25.4 bits [6]. Since the attribute space for Garvan ES1 is of size 46 bits, the region of experience for Garvan $K1$ is $2^{-20} \approx 10^{-6}$ of the total attribute space. This gain of 20 bits is more than half of the maximum possible gain for Garvan ES1. There are nearly 4000 distinct cases, so the region of experience is at least 12 bits large, so the maximum gain is $46 - 12 = 34$ bits.

This result suggests that a plausible strategy for inducing K is to first induce $K1$, proceeding further only if $K1$ is not strong enough. A very simple algorithm suffices to compute $K1$.

Algorithm 32. Compute $K1$ from a set of cases C.

Construct an above the diagonal triangular array A of integers each of whose rows and columns is indexed by a member of one of the value sets of one of the variables underlying C. Initialise this matrix to 0.

For each member c of C, increment all the cells in A which correspond to pairs of values of variables in c.

Each cell of A now contains the number of instances the associated variable values co-occur in C.

Each variable value $x = v$ selects a set of cells of A corresponding to all other variable values. If all but one of the cells associated with a given value is zero, with $x' = v'$ indexing the nonzero cell, then $x = v \rightarrow x' = v'$ is a partial functional dependency.

Algorithm 32 requires an array which is square in the aggregate cardinality of the value sets (which is about 100 for Garvan ES1). It would be practicible to extend the algorithm to three dimensions to compute $K2$, since the size of the corresponding matrix would be cubic in the aggregate cardinality of the value sets.

We now know that it is possible, in at least one case, to economically induce a small, strong set of PFDs. It remains to see what effect K has on the inflation problem.

An experiment was run using the algorithm resulting from Theorem 20 (table → tree), using a slightly different attribute selection procedure, applied to the 653-rule decision table resulting from applying the algorithm derived from Theorem 10 to the Garvan ES1 rule set, then selecting only those rows which were consistent with all of the 9805 cases [4]. In this experiment, the algorithm generated a tree with 46,378 leaves, of which 4562 were consistent with Garvan $K1$, and 985 were consistent with at least one case.

We can conclude from this section that the method of representing K as a set of partial functional dependencies is plausibly a practical approach to the inflation problem. The practicality issue is addressed in Section 6 below.

5. Further properties of K

The set K of constraints was introduced as a heuristic designed to control the inflation problem when translating propositional classification systems from one form to another. We observed in passing that another use for K was as a filter to detect cases that were either spurious or, if valid, at least outside the experience on which the classification expert system is based.

If we take the use of K as a filter seriously, and stipulate that the expert system is a partial function defined only on the subset of the attribute space covered by K, we can incorporate K into the translation theorems obtaining a corresponding set of translation procedures which is likely to be much less susceptible to inflation.

Proposition 33. *Given a conjunction A of elementary propositions and a case C associated with the same variable set as A; either A is inconsistent, A and C are inconsistent, or $A \& C = C$.*

Proof. By Proposition 17, a conjunction of elementary propositions is either inconsistent or a choice from the sets of elementary propositions associated with a subset of the variables. A case is a choice from the complete set of variables. Therefore in the conjunction of an arbitrary consistent conjunction A of elementary propositions, for each elementary proposition in A there is an elementary proposition in C associated with the same variable. If they are distinct, A and C are inconsistent with each other. Otherwise $A \& C = C$ by the Boolean algebra identity $a \& a = a$ applied to the elementary propositions in the conjunction. □

Proposition 34. *For any consistent set K of partial functional dependencies and any case C covered by K, $K \& C = C$.*

Proof. K can in principle be represented in disjunctive normal form in elementary propositions. For each disjunct D, Proposition 33 gives either $C\&D = C$ or C is inconsistent with D. Since C is consistent with K, at least one of the disjuncts must be consistent with C. The result follows from the Boolean algebra identity $A \vee A = A$. □

We now modify Theorems 10, 20 and 28.

Theorem 35 (Rules → table K). *An acyclic rule system defined on the region of the attribute space covered by a set K of partial functional dependencies is equivalent to a decision table defined on the same region of the attribute space.*

Proof. Theorem 10 proceeds by unfolding the set of rules, removing intermediate propositions, starting from intermediate propositions which are the consequents of rules having only elementary propositions in their antecedents. If a particular rule is of the form $p \rightarrow q$, then the application of the rule to a case c follows the derivation process:

(1) $c,\ p \rightarrow q$.

(2) $c,\ c \,\&\, p \rightarrow q$.

(3) If c is consistent with p: $c, c \rightarrow q$ (by Proposition 33).

(4) $c,\ c \rightarrow q,\ q$.

If q is an intermediate proposition, the process of unfolding in Theorem 10 creates a single proposition which is the only proposition whose consequent is the intermediate proposition, and similarly for its negation. By Proposition 34, step (2) in the derivation can be replaced by

(2′) $c, c \,\&\, K \,\&\, p \rightarrow q$.

If c is consistent with p, then K is consistent with p, for any c, by the definition of K. If the antecedent p is expressed in disjunctive normal form, then any disjunct inconsistent with K can be removed, as step (3) of the derivation will never succeed.

With this modification, the process of unfolding ultimately results in a decision table all of whose rows are consistent with K. □

An algorithm derived from Theorem 35 will prune the rows of the decision table as they are constructed, so will arrive smoothly at the final decision table consistent with K.

Theorems 19 and 20 were concerned with a complete propositional system, which was able to classify any possible case in the attribute space. Completeness as defined in Definitions 8 and 15 is no longer relevant, since we are looking at decision objects which are deliberately incomplete. This leads to the following definition of completeness with respect to a set of constraints.

Definition 36. A decision object P is *complete with respect to a set of partial functional dependencies K* if P classifies every case covered by K.

Proposition 37. *If $N(P)$ is the disjunction of cases not classified by P, then P is complete with respect to a set of partial functional dependencies K if $N(P)$ is inconsistent with K.*

Proof. By inspection. □

Proposition 37 may not lead to computationally feasible algorithms in all circumstances. However, if $N(P)$ can be represented in disjunctive normal form with m conjuncts, then the test for competeness with respect to K is no worse than testing whether m cases are in the region of experience defined by K. One would expect that the leaf propositions in a decision tree would tend to be relatively small (note that don't care cells can be eliminated by the Boolean algebra identity $a\,\&\,\mathit{true} = a$). A leaf proposition in disjunctive normal form with say 10 conjuncts per disjunct, having say 5 disjuncts, requires computation of 10^5 conjuncts to convert its negation to disjunctive normal form, which is tolerable. Most of these conjuncts would probably either be inconsistent or subsumed by others, so that the resulting expression would likely not have an excessive number of disjuncts. One would expect that there would be problems where the concept could be applied.

Definition 36 leads to modifications of Theorems 19 and 20.

Theorem 38 (Tree → table K). *A decision tree defined on the region of the attribute space covered by a set K of partial functional dependencies is equivalent to an unambiguous decision table defined on the same region of the attribute space, which is complete with respect to K if the tree is.*

Proof. Follows Theorem 19, except that only paths through the decision tree consistent with K are included in the decision table, since if a path is inconsistent with K, no valid case will be consistent with it.

If the tree is complete with respect to K, then for every case c covered by K, there is one path through the tree consistent with c. This path $P(c)$ is consistent with K by definition of K, so is included in the table. Thus every case c covered by K is consistent with at least one row of the table. □

Theorem 39 (Table → tree K). *An unambiguous decision table system defined on the region of the attribute space covered by a set K of partial functional dependencies is equivalent to a decision tree defined on the same region of the attribute space, which is complete with respect to K if the table is.*

Proof. Parallels that of Theorem 20. Replace part (3) in the induction step with

(3′) For each v in the value set of a such that the path proposition in the tree down to a $P(a, v)$ is consistent with K, create:
- An arc whose source is the deciding node created and whose arc proposition is $a = v$.
- A decision table $T(a, v)$ with one row derived from each row r of T in which the cell associated with a is consistent with $a = v$, and such that $P(a, v)\,\&\,r$ consistent with K, by removing the cell associated with a.

This proves that the tree is equivalent to the table on K, since no branch pruned is consistent with K, nor is any row excluded from the $T(a, v)$. The tree is therefore complete with respect to K if the table is. □

Finally, we modify Theorem 28 (rdr-tree → table), giving

Theorem 40 (rdr-tree → table K). *A ripple-down rule tree is equivalent with respect to K to an unambiguous decision table.*

Proof. The decision table is the union of all rdr path propositions which are consistent with K. The remaining observations from the proof of Theorem 28 apply. □

We have now achieved a practical set of translation procedures for decision objects taking into account that they are partial functions on their attribute space. The issue of practicality is canvassed below.

6. Discussion

The main result of this paper has been the solution of the problem of inflation in the translation of decision objects from one form to another. The key idea has been the characterisation of the region of experience using a set of constraints expressed as partial functional dependencies.

These problems are conceptually quite simple. Their difficulty arises from their origin in exponential data structures and exponential algorithms. It is easy to construct worst-case examples where the computations are very long, so they depend for their utility on the typical case turning out to be tractable, in much the same way as the simplex algorithm in linear programming. The procedures have been tested on a large problem and shown to be practicable, but one would not be able to routinely recommend them until they had been used by a wide variety of people on a wide variety of problems and intractable cases did not arise in practice—again like the simplex algorithm—which is well beyond the scope of a single project.

We can, however, consider the characteristics of situations where the procedures would fail. There are two things which could go wrong:

- the set of PFDs comprising K could have few rules with a small number of antecedents, instead very many rules with a large number of antecedents; or
- the use of K could fail to prune the decision objects in the translation process.

The former would be a problem both because the cost of computing K increases rapidly with the number of antecedents in the rules, and partly because of the cost of using a very large set of rules with a large number of antecedents. A system with an impracticable K would be extremely complex: it would be large, else the exponential algorithms would not reach their practical limits (a six-attribute decision table can be exhaustively analysed); and it would lack simple relationships so would be difficult for humans to either understand or to gather enough data for a statistically reliable decision object induction.

The latter mode of failure requires that the inflation of the decision objects be highly anti-correlated with the set of constraints. This would be surprising, because the decision object is built from the relationships of the case elementary propositions to the classification elementary propositions and the set of constraints is built from the relationships of the case elementary propositions among themselves. One would think that a system where the two were strongly correlated would be noticeably peculiar.

This highly informal argument indicates that the results of this paper are likely to be of wide applicability.

Solving the inflation problem with a set of constraints suggests a number of problems which will be the subject of further research.

First, the study of the set of constraints itself: estimation of its size, simplifying it, etc., which can likely make use of the techniques of constraint logic programming (e.g., [14]). Second, results in the theory of propositional expert systems can be revisited and improved, for example, the computational stability of expert systems as reported in [3]. In that work, the chief problem was to analyse a decision object in the form of a decision table to identify situations, where small changes in attribute space make large changes in classification space, using essentially a Hamming distance measure on attribute space. The results could be improved by considering only errors which remain within the region of experience.

Of course, use of the set of constraints as a filter can improve the brittleness of a wide range of expert systems, improving results such as that of Webb and Wells [23], which used method based on classification. In particular, there are problems such as described by Davidsson [9] in which a good characterisation of the region of experience is crucial. In that work, problems such as design of a coin-sorting machine are considered. In Europe, one often finds coins from other countries in a set of coins to be sorted. It becomes essential to recognise that a coin is foreign and to reject it while sorting the coins of a particular country. Use of partial functional dependencies should be able to improve Davidsson' results. Further, in the collection of data describing the region of experience one could also obtain data describing a range of foreign coins, and so also have a set of cases which is explicitly outside the desired region of experience, which could improve the estimates of K.

Edwards et al. [10] has investigated the problem of *prudence* in an expert system. The type of system concerned is an automated medical pathology laboratory system, which produces clinical interpretations of samples using the analysis machine results and a small amount of descriptive information about the patient. (The system, called PIERS, is a descendant of Garvan ES1.) In the application, it is a legal requirement that all clinical interpretations be signed by an appropriate specialist. This requirement represents a significant residual human involvement.

The proposal of Edwards et al. is for the expert system to maintain records of all the cases it processes and to append to an interpretation how different the present case is from other cases previously processed. Cases very similar to previous cases can be assumed to have very reliable interpretations, while the more different a case is from previous cases, the closer checking the interpretation should have. Edwards et al.'s approach is to use the range of values for particular variables as a measure of similarity. We speculate that an incremental computation of a set of partial functional dependencies might give a more general and more reliable measure. In particular, we would investigate including all partial functional dependencies, no matter how weak their support, so long as they have 100% confidence. A new case can either increase the support of a dependency or break it, so that earlier estimates would tend to overconstrain the region of experience, which would be gradually widened as the system was used. Each case violating a constraint would be subject to special scrutiny.

Finally, note that the characterisation of the region of experience by a set of constraints differs fundamentally from clustering methods: the set of constraints identifies the boundary of the region, while clustering methods generally identify the centres of regions of interest. It might be profitable to investigate the interaction of the two kinds of method.

References

[1] R. Agrawal, R. Srikant, Mining Generalized Association Rules VLDB'95, Morgan Kaufmann, Los Altos, CA, 1995, pp. 407–419.

[2] K.R. Apt, H.A. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Database and Logic Programming, Morgan Kaufmann, Los Altos, CA, 1988, pp. 89–148.

[3] R.M. Colomb, Computational stability of expert systems, Expert Systems with Applications 5 (1992) 411–419.

[4] R.M. Colomb, Y.-P. Chen, Use of partial functional dependencies to make practical approximate translations among forms of propositional expert systems, in: A. Sattar (Ed.), Proc. 5th Australian Joint Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 1342, Springer, Berlin, 1997, pp. 167–176.

[5] R.M. Colomb, C.Y. Chung, Strategies for building propositional expert systems, Internat. J. Intelligent Systems 10 (1995) 295–328.

[6] R.M. Colomb, J. Sienkiewicz, Analysis of redundancy in expert systems case data, in: Proc. 8th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1995, pp. 395–402.

[7] P. Compton, R. Jansen, A philosophical basis for knowledge acquisition, Knowledge Acquisition 2 (1990) 241–257.

[8] B.J. Cragun, H.J. Steudel, A decision-table-based processor for checking completeness and consistency in rule-based expert systems, Internat. J. Man-Machine Studies 26 (1987) 633–648.

[9] P. Davidsson, Learning characteristic decision trees, in: Proc. 8th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1995, p. 579.

[10] G. Edwards, B.H. Kang, P. Preston, P. Compton, Prudent expert systems with credentials: managing the expertise of decision support systems, Internat. J. Biomedical Computing 40 (1995) 125–132.

[11] K.A. Horn, P. Compton, L. Lazarus, J.R. Quinlan, An expert computer system for the interpretation of thyroid assays in a clinical laboratory, Australian Computer J. 17 (1985) 7–11.

[12] J. Jaffar, M.A. Maher, Constraint logic programming: A survey, J. Logic Programming 19/20 (1994) 503–581.

[13] Z. Pawlak, Rough Sets: Theoretical Aspects of Reasoning about Knowledge, Kluwer Academic, Dordrecht, 1991.

[14] M.J. Maher, Constrained dependencies, Theoret. Comput. Sci. 173 (1997) 113–149.

[15] J.R. Quinlan, Semi-autonomous acquisition of pattern based knowledge, in: J.E. Hayes, D. Michie, Y.-H. Pao (Eds.), Machine Intelligence 10, Ellis Horwood, London, 1982, pp. 159–172.

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

[17] T. Sato, Equivalence-preserving first order unfold/fold transformation systems, Theoret. Comput. Sci. 105 (1992) 57–84.

[18] H. Seki, Unfold/fold transformations of stratified programs, in: G. Levi, M. Martelli (Eds.), Logic Programming: Proc. 6th International Conference (Lisbon), MIT Press, Cambridge, MA, 1989, pp. 554–568.

[19] K. Shwayder, Conversion of limited-entry decision tables to computer programs—A proposed modification to Pollack's algorithm, Comm. ACM 14 (1971) 69–73.

[20] K. Shwayder, Extending the information theory approach to converting limited-entry decision tables to computer programs, Comm. ACM 17 (1974) 532–537.

[21] J.D. Ullman, Principles of Database and Knowledge-Base Systems, Vol. 1, Computer Science Press, Rockville, MD, 1988.

[22] J.D. Ullman, Principles of Database and Knowledge-Base Systems, Vol. 2, Computer Science Press, Rockville, MD, 1989.

[23] G.I. Webb, J. Wells, Recent progress in machine-expert collaboration for knowledge acquisition, in: Proc. 8th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1995, pp. 291–298.