THE COMPUTATIONAL COMPLEXITY OF ASSUMPTION-BASED TRUTH MAINTENANCE SYSTEMS

by

Gregory M. Provan

Technical Report 88-11

July 1988

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5 Canada

THE COMPUTATIONAL COMPLEXITY OF ASSUMPTION-BASED TRUTH MAINTENANCE SYSTEMS

Gregory M. Provan[1]

Technical Report 88-11

July 1988

Department of Computer Science
University of British Columbia
Vancouver, BC
Canada V6T 1W5

Abstract

We define the complexity of the problems that the Assumption-Based TMS (ATMS) solves. Defining the conjunction of the set of input clauses as a Boolean expression, it is shown that an ATMS solves two distinct problems: (1) generating a set of minimal supports (or label) for each database literal; and (2) computing a minimal expression (or set of maximal contexts) from the set of minimal supports. The complexity of determining the set of minimal supports for a set $\mathbf{x}$ of literals with respect to a set X of clauses is exponential in the number of assumptions *for almost all Boolean expressions*, even though a satisfying assignment for the literals occurring in X can be found in linear time. Generating a minimal expression is an NP-hard problem. The ATMS algorithms can be used with many control mechanisms to improve their performance for both problems; however, we argue that manipulating the label set (which is exponential in the number of assumptions) requires considerable computational overhead (in terms of space and time), and that it will be infeasible to solve moderately large problems without problem restructuring.

[1]The author completed this research with the support of a Scholarship from the Rhodes Trust, Oxford, and of the University of British Columbia Center for Integrated Computer Systems Research, BC Advanced Systems Institute and NSERC grants to A.K. Mackworth.

1 INTRODUCTION

1.1 Background

A Truth Maintenance System (TMS) is an AI tool widely used for database updating and consistency maintenance tasks. Examples of areas in which it has been used include qualitative process theory—[10]; circuit analysis— [12]; analog circuit design—SYN [11]; temporal reasoning [37]; and vision— [17], [4]. Despite its wide use, it is not well understood in terms of the problems it solves, its semantics or its complexity.

The uniqueness of a TMS stems from maintaining records of the origins of labels assigned to database facts (dependency records), and subsequently using those dependency records to prune the search space and perform database updating. The TMS was originally designed as an intelligent cache and a means of expediting dependency directed backtracking ([34], [15]) during search for a consistent labelling of the fact set. Subsequent implementations (e.g. by de Kleer [6] and McDermott [24]) extended these facilities to include maintenance of dependency records not just for one consistent database labelling (or context) but for multiple contexts. All implementations use dependency records to maintain consistent labellings and reduce the updating required when new data is added to the database. Systems which do not record dependencies must completely be rerun to ensure consistency when new data is added.

In this paper we focus on one implementation of a TMS, the ATMS. An ATMS simultaneously determines all minimal "solutions" given a set of clauses. The ATMS was cited [7] as solving several drawbacks inherent in previous TMS implementations, i.e. JTMSs [22]. For example, the ATMS almost never needs to backtrack when an inconsistency is detected, whereas a JTMS must backtrack to recover from inconsistencies, which can be a computationally costly operation.

However, the ATMS requires a large computational overhead relative to a JTMS. This paper identifies and explicates such computational overheads by precisely defining the problems ATMSs solve and the worst-case complexity of those problems. We show that the ATMS solves Boolean algebra problems defined in both theorem proving and switching theory, and use the notation from both areas in our formalisation.

1.2 Overview

This paper is organised as follows. In Section 2, we define our notation and the tasks the ATMS solves in terms of this notation. In Section 3 we derive the complexity results for the ATMS tasks as defined in Section 2. We review related work in Section 4, and discuss the impact of these complexity results on the ATMS, when viewed as a reasoning tool, in Section 5.

2 PRELIMINARIES

2.1 Notation

We use a propositional language which contains a finite set L of propositional symbols and the connectives $\vee$, $\wedge$ and $\neg$, defining the connective $\supset$ (or $\Rightarrow$) in terms of $\vee$ and $\neg$ in the usual way. A propositional literal is a propositional symbol or its negation. A *clause* is a finite disjunction of propositional literals, with no repeated literals. $X = \{X_1, ..., X_n\}$ is a set of *facts*, which we define as the set of input clauses. Upper-case, subscripted X's represent facts and lower-case subscripted x's represent literals. For a fact of the form $\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_m \vee x$, x is called the *consequent* and $x_1, ..., x_m$ the *antecedents*. A fact with $m = 0$ is called a *premise*.

A TMS records dependencies among a set of clauses to maintain a consistent valuation, or assignment of set of labels to literals and/or clauses. For the ATMS such labels consist of sets of distinguished literals called assumptions. A literal x_i is dependent on literal x_j if the valuation of x_i changes as the valuation of x_j changes. A TMS uses such dependency information to rule out regions of the search space and to efficiently indicate necessary database changes when contradictions are discovered.

The input to an ATMS consists of a set $X = \{X_1, ..., X_l\}$ of Horn-clause *facts*, referred to in [6] as justifications. A Horn-clause is a clause with at most one unnegated literal. For example, a Horn-clause X_i can be written as $\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee ... \vee \overline{x_k} \vee x$, $k \geq 0$.[2] We call the conjunction of the X_i's a Boolean polynomial function[3] F, i.e. $F = \bigwedge_{i=1,...,l} X_i$. We note that there may be many other polynomials F' which also compute such a partially specified Boolean function F, where a polynomial F' computes F if $F'(x) = F(x)$ for $x \in F^{-1}(\{t, f\})$. We define the cost of F as the number of clauses in F. A minimal polynomial $\mathcal{F}$ is a polynomial such that $\mathcal{F}$ computes F and no polynomial computing F has cost smaller than $\mathcal{F}$.[4] By defining this Boolean polynomial F, we explicitly show how the ATMS can be thought of as a means of designing an optimal circuit: given a Boolean polynomial F which represents a circuit, the ATMS can find an optimal polynomial (or circuit) $\mathcal{F}$ using some optimisation criterion, such as minimising the number of AND-gates.

Using the definition of [33], a prime implicate[5] of a set X of clauses is a clause π such that

[2] We represent a fact denoting an implication, i.e. one written as $x_i \supset x_j$ in the TMS literature, as $\overline{x_i} \vee x_j$ by convention.

[3] This is standard conjunctive normal form (CNF), the dual representation of traditional disjunctive normal form (DNF) Boolean functions.

[4] Defining the cost of the minimal polynomial in this way explicitly assumes that the "solution" is represented in terms of the smallest number of clauses (in ATMS terminology assumption sets) which explain the given evidence. There are several other ways of defining cost, such as the total number of literals (in ATMS terminology assumptions) contained in the minimal polynomial.

[5] The dual to prime implicate (in Boolean algebra) is called a prime implicant. We use the prime implicate terminology to avoid confusion between the dual representations.

- $X \models \pi$, and
- For no proper subset π' of π does $X \models \pi'$.

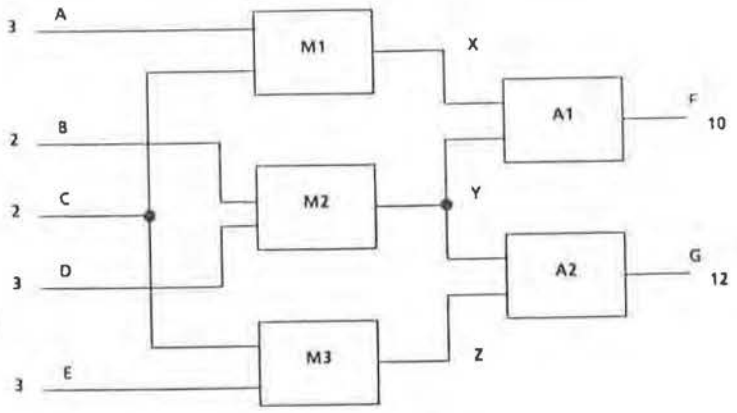
ξ_j is a *support clause* for x_i with respect to X iff $X \not\models \xi_j$, $x_i \cup \xi_j$ does not contain a complementary pair of literals (i.e. both x_j and $\overline{x_j}$), and $X \models x_i \cup \xi_j$. ξ_j^* is a minimal support for x_i with respect to X iff no proper subset of ξ_j is a support for x_i with respect to X. In [6] the set of minimal supports for any literal x_i is referred to as the label for x_i. $\xi^*(x_i)$ is the set of minimal supports for x_i with respect to X, and Ξ^* is the set of minimal supports for $\mathbf{x}$. We show the close relationships between support clauses and prime implicates, as demonstrated in [33], in the next section.

2.2 A Logical Reconstruction of the ATMS

The ATMS records dependencies in terms of a distinguished subset $A \subseteq \mathbf{x}$ of literals called *assumptions*. We refer to an *environment* E as a set of assumptions and a *context* C as the set of literals derivable from the assumptions in E given the facts. We assume, as in [6], that the Problem Solver generates clauses for the ATMS and employs control mechanisms (e.g. consumers[6]) to ensure problem solving efficiency.

Assumptions are the fundamental literals with which the derivation of all other literals are recorded. For example, consider the well-known circuit consisting of multipliers M_1, M_2 and M_3 and adders A_1 and A_2, as shown in Figure 1. The output at F is 10 instead of 12,

Figure 1: Circuit with faulty components



meaning that some combination(s) of M_1, M_2, M_3, A_1 and A_2 is(are) faulty. There are a total of $2^5 = 32$ combinations of faulty components. Taking observations at points like X, Y or Z narrows the set of diagnoses consistent with the observations and guides future decisions about where to make further readings. A solution consists of a set of faulty multipliers and adders which explain all the observations.

[6] See [9] for a full description of consumers.

A system incorporating an ATMS, GDE [12], can simultaneously find all sets of faulty circuit components which explain the observations for such circuits. In GDE, the ATMS identifies hypothesised sets of circuit components whose faulty behaviour could cause discrepancies between predicted and observed circuit measurements. For this network diagnosis problem, assumptions can be:

- each component is working, represented by M_1, M_2, etc. in Figure 1;
- input values, e.g. $A = 3$, $B = 2$, etc.

Facts containing an unnegated literal, e.g. $\overline{x_1} \vee \overline{x_3} \vee x_5$, have an antecedent (which can be a set of assumptions, as in $\overline{x_1} \vee \overline{x_3}$) justifying a consequent x_5, which cannot be an assumption. We also call a derived literal the consequent of a fact. In the network diagnosis example, facts could be as follows:

$$\overline{M_1} \vee (X = 6)$$
$$\overline{M_2} \vee (Y = 6)$$
$$\overline{A_1} \vee (F = 6).$$

The first of these facts means that if M_1 is working, the reading at point X in the circuit should be 6.

The ATMS's operation consists of two distinct phases, label manipulation and interpretation construction, which we describe below.

Label Manipulation: In the label manipulation phase, for each literal the ATMS maintains a *label*, which is a set of environments in which the literal can be proven. Each environment consists of the greatest lower bound (GLB) of assumption sets. In logical terms, each environment in the label for x_i is defined in terms of a minimal support clause ξ_k^* for x_i, and is given by

$\xi_k^* = \{\bigwedge_j A_j$ for some $j \mid (\bigvee_j \overline{A_j})$ is a minimal support clause of ^{X}

The label for derived literals is assigned through *assumption set propagation*, based upon the set of justifications for the derived literal. Assumptions typically[7] have themselves in the label. The label for derived literals is assigned by combining the labels of the antecedents of the derived literal. In simplest terms, the label update algorithm performs a set union of the labels. If a literal x has k justifications, and justification i has label $L_i = \{L_{i1}, L_{i2}, ..., L_{im}\}$, where each L_{ij} is an assumption set, the label for x is given by $\bigcup_i \bigcup_j L_{ij}$, noting that each assumption set in the resultant label is recorded in terms of its GLB. Labels are assigned to a derived literal by taking the set union of all combinations of labels for its antecedents and then using subsumption to ensure the new label is represented in GLB form. Inconsistencies are removed by identifying the inconsistent sets of assumptions (nogoods) and removing these sets and their supersets from all labels.

[7] See [6], [8], [9] for cases in which this is not true.

For example, if we have $\overline{x_1} \vee \overline{x_3} \vee x_5$, (or $x_1 \wedge x_3 \supset x_5$), and x_1 and x_3 have labels $\{\{A,B\}, \{B,C\}\}$ and $\{\{A\}, \{D,E\}\}$ respectively, then x_5 is assigned the label $\{\{A,B\}, \{B,C,D,E\}\}$.

In logical terms, the label generation algorithm actually calculates the set of prime implicates Π for X. From these prime implicates the support sets for each literal are derived. In the switching theory literature, many methods of generating prime implicants, the dual to prime implicates, exist. Examples include the methods by Quine and McCluskey [23], Tison [35] and Hwang et al. [18]. Reiter and de Kleer [33] have characterised the ATMS in terms of prime implicates and support sets, and we quote four Lemmas concerning their relationship and refer the reader to [33] for their proofs.

Lemma 1 *Suppose X is a set of clauses and X_i is a clause. If ξ is a minimal support clause for X_i with respect to X, then there is a prime implicate π of X such that $\pi \cap X_i \neq \emptyset$ and $\xi = \pi - X_i$.*

Lemma 2 *Suppose X is a set of clauses and X_i is a non-empty clause. If π is a prime implicate for X such that $X_i \subseteq \pi$, then $\pi - X_i$ is a minimal support clause for X_i with respect to X.*

Lemma 2 states that once the set of prime implicants have been derived, the minimal support clauses for all the literals in X can be determined.

The derivation of the prime implicates $\Pi = \{\pi_1, ..., \pi_l\}$ enables the polynomial F to be represented in terms of the prime implicates, i.e. $F_\Pi = \bigwedge_i \pi_i$, using the following lemma:

Lemma 3 *For a set X of clauses, X and its set of prime implicates Π are logically equivalent in the sense that if $X_i \in X$, then $\Pi \models X_i$, and if $X_i \in \Pi$, then $X \models X_i$.*

In the terminology of this paper, F_Π computes F.

If we restrict the clauses for which we determine minimal support clauses to literals, then the prime implicates can be derived from the minimal support clauses or vice versa, which is not true in general.

Lemma 4 *Suppose X is a set of clauses and x is a literal. ξ is a minimal support clause for x with respect to X iff there is a prime implicate π for X such that $x \in \pi$ and $\xi = \pi - x$.*

A major difference between the ATMS and other methods of generating prime implicates is that the ATMS generates them incrementally. That is, if it computes the set of prime implicates Π for X, it can use this set Π to compute the new set of prime implicates as new clauses are introduced to produce a new clause set $X' \supset X$. We now explain in more detail how this is done. Consider adding a new clause X'_k whose consequent is a derived literal of the original set of literals x. In de Kleer's terminology, assumption set propagation is conducted on this literal using the new clause X'_k, and the new label is compared with

the existing label. By this comparison[8] a *complete and minimal* label is constructed. By complete we mean each environment which should be present in the label is actually there. By minimal we mean that no two environments in the label are supersets of one another. If there are any changes to the existing label for literal x, the effects of these changes are propagated throughout the label set for all literals which are affected by the label updating of x.

For example, consider the following set of clauses, which we represent both as implications and in typical clausal form:

$$\begin{array}{lll} x_1 \wedge A_1 \supset x_2 & & \overline{x_1} \vee \overline{A_1} \vee x_2 \\ x_2 \wedge A_2 \supset x_3 & & \overline{x_2} \vee \overline{A_2} \vee x_3 \\ x_1 \wedge A_3 \supset x_4 & & \overline{x_1} \vee \overline{A_3} \vee x_4 \\ x_4 \wedge A_4 \supset x_5 & & \overline{x_4} \vee \overline{A_4} \vee x_5 \\ x_3 \wedge A_5 \supset x_5 & & \overline{x_3} \vee \overline{A_5} \vee x_5 \end{array}$$

Additionally, x_1 and x_6 are premises.

The support sets (or labels) for the literals x_2, x_3, x_4 and x_5 are given by:

LITERAL	LABEL
x_2	$\{A_1\}$
x_3	$\{A_1, A_2\}$
x_4	$\{A_3\}$
x_5	$\{\{A_3, A_4\}, \{A_1, A_2, A_5\}\}$

Consider what happens when a nogood corresponding to the conjunction of x_2 and x_3 is discovered. This nogood consists of the conjunction of the labels for x_2 and x_3, i.e.

$$\begin{array}{rcl} nogood & = & (A_1) \wedge (A_1 \wedge A_2) \\ & = & (A_1 \wedge A_2) \end{array}$$

All supersets of the nogood are removed from the labels of all literals. Hence, x_3 will now have no label, since its old label is a nogood. The new support sets (or labels) for the literals x_2, x_3, x_4 and x_5 are given by:

LITERAL	LABEL
x_2	$\{A_1\}$
x_3	$\{\}$
x_4	$\{A_3\}$
x_5	$\{A_3, A_4\}$

[8] See [6] for full details.

If the clause $\overline{x_2} \vee \overline{A_6} \vee \overline{x_4}$ is added, a nogood is created since both x_4 and $\overline{x_4}$ have support. This nogood consists of the conjunction of the labels for x_4 and $\overline{x_4}$, i.e.

$$\begin{aligned} nogood &= (A_3) \wedge (A_1 \wedge A_6) \\ &= (A_1 \wedge A_3 \wedge A_6) \\ &= \{A_1, A_3, A_6\} \text{ in alternative format.} \end{aligned}$$

No labels are affected by this nogood.

Now, if another new clause whose consequent is x_4 is added, i.e. $\overline{x_6} \vee \overline{A_7} \vee x_4$, the label for x_4 now becomes $\{\{A_3\}, \{A_7\}\}$. The revised label set is

LITERAL	LABEL
x_2	$\{A_1\}$
x_3	$\{\}$
x_4	$\{\{A_3\}, \{A_7\}\}$
x_5	$\{\{A_3, A_4\}, \{A_4, A_7\}\}$

As stated by de Kleer [6], the *task of the ATMS labelling procedure* is to guarantee the consistency, completeness and minimality of the label set (support clauses) of each literal with respect to the literals. This is ensured by generating the set of minimal support clauses for each literal in **x**.

The ATMS explores multiple solutions simultaneously, implicitly representing each solution with a *context*. A derived literal is contained in a context if the assumptions in at least one of the environments in its label are a subset of that context. Multiple labels for a literal x_i indicate that x_i is present in multiple contexts. However, this procedure does not explicitly calculate contexts, but manipulates labels. Contexts are explicitly computed by an interpretation construction algorithm.

Interpretation Construction: In the second phase of operation, the ATMS constructs the *interpretations* to explicitly determine the set of "solutions" or maximal contexts. An *interpretation* is the smallest set of assumptions from which all literals in the context are derivable, and a context is maximal if it has no consistent superset contexts or no supersets (when all assumptions are consistent).

In logical or switching theory terminology, the ATMS computes the minimal polynomial $\mathcal{F}$ for F, using the set of prime implicates which comprise F_Π. To outline how this terminology relates to ATMS problems, we give two examples of the correspondence between minimal polynomial and "solutions".

- In fault diagnosis using GDE [12], there is a correspondence between the system components and circuit behaviour. Specifically, hypotheses consisting of non-functioning sets of components can explain the observed output of the electrical circuit. Given the set of all possible readings of circuit behaviour, the hypotheses, loosely speaking,

"cover" (in set covering terms) the actual readings of circuit behaviour. X^* corresponds to a largest set of components which are not faulty, or represented as its dual, a smallest set of faulty components (candidates). GDE identifies all such candidates. The minimal polynomial consists of a disjunction of assumption sets, each of which is a conjunction of assumptions. Each assumption set identifies a candidate.

- In the figure recognition task described in [29], the task is to identify instantiations of complete figures (and of almost-complete figures) from a set of rectangles randomly arranged in a scene. In this domain, a given set of rectangles may overlap in such a way to correspond to a complete puppet. X^* corresponds to a largest set of rectangles which can be interpreted as a complete or almost-complete puppet figure. Similar to the previous example, each conjunction of assumptions in the minimal polynomial corresponds to a complete, minimal interpretation of a puppet figure.

ATMS Facilities:
We summarise the facilities offered by the ATMS as follows:

Label maintenance: The ATMS generates and maintains a consistent label set for each database literal, given changes to the database.

Query processing: The ATMS can answer the query: identify the label set for literal x?

Solution maintenance: The ATMS can simultaneously generate all minimal solutions (i.e. the minimal polynomial) and then incrementally update this solution set.

3 ATMS COMPLEXITY RESULTS

We now examine the complexity of the problems underlying the ATMS. We shall prove several NP-completeness and NP-hardness results. To show a problem to be NP-complete, we must show that it is in NP, and that an NP-complete problem transforms to this problem in polynomial time. Loosely speaking, NP-hard problems are at least as hard as NP-complete problems, but not provably in NP. An example of an NP-hard problem is the optimisation version of a well-known NP-complete problem, the Travelling Salesman problem [25].

We derive the ATMS complexity result by defining ATMS processing as solving the following two problems:

Generation of minimal supports: Given a set X of facts (i.e. Horn clauses), determine the set of minimal supports Ξ^* for the set $\mathbf{x}$ of literals in X.

Interpretation Construction: Given the set of minimal supports Ξ^* for $\mathbf{x}$ with respect to a fact set X, find the minimal polynomial $\mathcal{F}$.

We now examine the complexity of each of these problems in turn.

3.1 Minimal Supports

To define the complexity of the minimal supports problem, we need the following lemmas:

First, we show that computing the existence of a minimal support clause ξ for a literal x with respect to a set of clauses X is as easy as computing the satisfiability of X.

Lemma 5 *X is unsatisfiable iff the only prime implicate is the empty clause $\square$.*

Proof: $\Longrightarrow$ Suppose X is unsatisfiable. Hence $X \models \square$ and hence we can choose $\square$ as an implicate, since it satisfies the definition of implicate. But since $\square$ subsumes all other clauses, it is not only a prime implicate, but the only prime implicate.

$\Longleftarrow$ Suppose $\square$ is the only prime implicate. By definition, $X \models \square$, and hence X is unsatisfiable. ■

From this lemma we have an obvious corollary:

Corollary 1 *X is satisfiable iff $\exists$ a non-empty prime implicate.*

We can now prove the following Lemma concerning existence of minimal support clauses using Lemma 1:

Lemma 6 *$\exists$ a minimal support clause ξ for a literal x with respect to a set of facts X, such that $\xi \cup x$ is a non-empty prime implicate of X, iff X is satisfiable.*

Second, the time complexity of determining satisfiability of a set of Horn clauses is given by Lemma 7 [14].

Lemma 7 *A satisfying assignment of valuations to the literals in a set X of Horn clauses can be determined in $O(n)$ time.*

Using these complexity results, we can show the linear complexity of determining the existence of a set of minimal support clauses with respect to a set X of Horn clauses.

Theorem 1 (Set of Minimal Supports) *Determining if there exists a set of minimal supports for a set $\mathbf{x}$ of literals with respect to a set X of Horn clauses is of $O(n)$ complexity.*

Proof: By Lemma 6, a set of supports for X exists iff X is satisfiable. This means that determining the existence of a set of minimal supports for X is as hard as determining if X is satisfiable. Determining the satisfiability of X is $O(n)$, and hence determining the existence of minimal supports for X must be $O(n)$. ■

However, even though finding if there exists a set of minimal support is of linear complexity, we can show that actually generating the set of minimal supports Ξ^* (or prime

implicates $\Pi(X)$) is intractable. This is because there can be an number of minimal support clauses (prime implicates) exponential in the number n of literals for a set X of input clauses.

First, for the propositional case we quote worst-case bounds on the number of prime implicates generated by a Boolean expression F, expressed both in terms of the number of variables and the number of clauses in F.

Lemma 8 describes the worst-case upper bounds on the number of prime implicates, expressed in terms of the number n of variables in the Boolean expression:

Lemma 8 *In the worst case, the number of prime implicates in a Boolean expression containing n variables is bounded by*

$$\Omega(3^n/n) \leq |\Pi| \leq O(3^n/\sqrt{n}).^9$$

Proof: See [5].

Lemma 9 describes the worst-case upper bounds on the number of prime implicates, expressed in terms of the number m of clauses in the Boolean expression:

Lemma 9 *In the worst case, the number of prime implicates in a Boolean expression containing m clauses is bounded by*

$$3^{\lfloor m/3 \rfloor} \leq |\Pi| \leq 2^m.$$

Proof: See [5].

Lemmas 8 and 9 express worst-case results, but Theorems 2 and 3 are much stronger results, as they provide upper and lower bounds for almost all Boolean expressions.[10] For computing $\Pi(X)$, the following complexity result has been shown.

Theorem 2 *Generating the set of prime implicates $\Pi(X)$ with respect to a set X of propositional clauses is of complexity exponential in the number n of literals for almost all propositional expressions F.*

Proof: From a result by Kuznetsov [20], the number of prime implicates in a minimal expression for almost all Boolean expressions F defined over n variables, is $\Theta(2^n)$. More specifically, as a constant $\varepsilon_n \to 0$, it is bounded by

$$\frac{(1-\varepsilon_n)2^n}{log\ n\ loglog\ n} \leq |\Pi^*(X)| \leq \frac{(1.6)\ 2^n}{log\ n\ loglog\ n},$$

[9] In the original paper, this lemma is stated in terms of finding bounds for a function $f(k)$, for $k \geq 1$, where $f(k)$ is defined by $f(k) = max\{\Pi(F) : F$ is in DNF with k conjuncts $\}$. Whether F is CNF or DNF is irrelevant to the worst-case bounds derived.

[10] A property is said to hold for almost all the functions of the algebra of logic if the proportion of functions of n variables which do not satisfy this property (among all the functions of n variables) tends to zero when $n \to \infty$. See [38] for details.

where $\Pi^*(X)$ denotes the collection of prime implicates in $\mathcal{F}$, as distinct from $\Pi(X)$, which denotes the collection on prime implicates in F_Π.

By definition of $\mathcal{F}$, $\mid \Pi^*(X) \mid \leq \mid \Pi(X) \mid \leq \mid \Gamma(X) \mid$. Hence, since $\mid \Pi(X) \mid = \Omega(2^n)$, then $\mid \Gamma(X) \mid = \Omega(2^n)$. To ensure minimality of each prime implicate in $\Pi^*(X)$, all implicates $\gamma \in \Gamma(X)$ must be tested for primality, which will require $\Omega(2^n)$ time. ■

This theorem states that almost all expressions have the same order of growth as for the most complex expression. From the previous results, it is simple to show that:

Theorem 3 (Set of Minimal Supports (Computation)) *Generating the set of minimal support clauses for a set* x *of literals with respect to a set X of clauses is of complexity exponential in the number n of literals for almost all propositional expressions F.*

Proof: Generating $\Pi(X)$ is $\Omega(2^n)$. Given $\Pi(X)$, $\Xi^*(X)$ can be computed in time $\ll \Theta(2^n)$.[11] Hence, generating $\Xi^*(X)$ is $\Omega(2^n)$. ■

Theorem 3 states that almost all Boolean expressions must have a number of minimal support clauses exponential in the number of literals. Hence, for almost all problems, determining a $\Theta(2^n)$ set of prime implicates will take $\Theta(2^n)$ time, and storing such a set of prime implicates will take $\Theta(2^n)$ space.

Note that the ATMS represents the minimal support in terms of a subset of the literals, the set of assumptions. Hence, the complexity results for the ATMS must be expressed in the number n of assumptions.

Consider a worst-case example of the exponential complexity of the set of minimal support, the parity problem. First the parity problem is defined as given in [9]: For an expression F of n variables $\{x_1, ..., x_n\}$, each of which can take on Boolean values, the parity of F is 1 if there is an odd number of variables set to 1, and 0 if there is an even number set to 1. The goal is to find an F with parity 1. Parity is defined recursively, calling p_i the parity for all variables up to and including variable x_i. Hence, if an implication like $(p_{i-1} = 0) \wedge (x_i = 1) \supset (p_i = 1)$ is written as $(\overline{p_{i-1} = 0}) \vee (\overline{x_i = 1}) \vee (p_i = 1)$, one obtains the boundary condition

$$p_{-1} = 0,$$

and for $i = 1, ...n$,

$$\begin{aligned}
&(\overline{p_{i-1} = 0}) \vee (\overline{x_i = 1}) \vee (p_i = 1),\\
&(\overline{p_{i-1} = 1}) \vee (\overline{x_i = 0}) \vee (p_i = 1),\\
&(\overline{p_{i-1} = 0}) \vee (\overline{x_i = 0}) \vee (p_i = 0),\\
&(\overline{p_{i-1} = 1}) \vee (\overline{x_i = 1}) \vee (p_i = 0).
\end{aligned}$$

This gives a total of $4n + 1$ input Horn clauses based on n variables $x_1, ..., x_n$ in addition to $n + 1$ added variables $p_0, ..., p_n$. Given this set of input clauses, it has been shown that:

[11] At present results are known only for queries q which are unit literals of conjunctions of literals. However, for this proof all that is necessary is an upper bound.

Lemma 10 *Let F be the parity function with n variables. Then even though a satisfying assignment can be found for F in $O(n)$ time, a minimal expression $\mathcal{F}$ consists of 2^{n-1} prime implicates of length n each.*

Lemma 10 was first proven by Lupanov [21]. It has been cited in the AI literature by McAllester [39], and discussed by de Kleer [9]. This Lemma states that there can exist an expression F with size $O(n)$ such that any minimal expression $\mathcal{F}$ which computes F must have $O(2^{n-1})$ size.

An obvious corollary is:

Corollary 2 *Generating the set of minimal support clauses for the parity problem is of complexity exponential in the number n of components.*

Proof: As shown in Lemma 10, there are 2^{n-1} implicates for the parity problem. Determining the set of prime implicates takes exponential time, as each implicate must be tested against the $2^{n-1} - 1$ other implicates to ensure minimality. ■

For this problem (and for others like it), it is impossible to know *a priori* which clauses generated by a prime implicate (or prime implicant) algorithm are prime. In fact, for the parity problem all are prime.

Obviously, parity is a worst-case example. However, Theorem 3 is a strong result and says that not only do there exist problems like parity with a set of minimal support clauses of size $\Theta(2^{n-1})$, but it also states that *almost all problems* will still be $\Theta(2^{n-1})$, where n is the number of ATMS assumptions.

As an example, consider the circuit diagnosis problem solved by GDE. Here, the size of the problem could be defined in terms of the sum of the number of circuit components and the number of circuit input/output values. An obvious question to explore is how the number of assumptions $|\mathcal{A}|$ relates to other measures of the "size" of the problem.[12] If $|\mathcal{A}|$ is much smaller than the size of the problem then these results will be less compelling. However, if $|\mathcal{A}|$ is larger than the size of the problem then the results will be very compelling.

In the circuit analysis domain as defined in GDE, assumptions include

- proper functioning of each circuit component, such as $\mathcal{W}_{A_1}, \mathcal{W}_{A_2}, \mathcal{W}_{M_1}, ...$, and
- values measured at various positions in the circuit, such as $\mathcal{V}_F, \mathcal{V}_G, ...$

The complexity of this problem is $\Theta(2^n)$, where n is the number of assumptions made ($|\mathcal{A}|$). In this case the number of assumptions made is at least as large as the size of the problem. This means that there are strict limitations on the size of circuits GDE can analyse, unless other methods are used, such as hierarchical analysis of the circuit, etc.

[12] The size of a problem is an attempt to define the number of components into which the problem can be broken down, and relate this to number of assumptions and other propositional literals the ATMS must manipulate. For the circuit analysis domain, a simple estimate of size is the number of components in the circuit, or the sum of the number of components and the number of measurement points.

Note that a Justification-Based TMS (specifically B-JTMS or LTMS) can find a satisfying assignment (if one exists) for the parity problem in $O(n)$ time. ATMS label manipulation introduces a significant computational overhead with respect to a JTMS, as it must generate a number of minimal support clauses exponential in the number of literals, irrespective of computing any satisfying assignment.

The ATMS stores the entire label set to avoid recomputing it every time it is needed. However, we can show that *incrementally updating* the label set given a new clause is as computationally expensive in the worst case of computing the label set from scratch. More precisely,[13]

Corollary 3 *Given the set of minimal support clauses Ξ^* for a set $\mathbf{x}$ of literals with respect to a set X of clauses, and a new clause X_{new} with some literals $\mathbf{x}_{new} \notin \mathbf{x}$, generating the set of minimal support clauses for the set $\mathbf{x} \cup \mathbf{x}_{new}$ of literals with respect to the set $X \cup X_{new}$ of clauses using Ξ^* is of exponential complexity in the worst case.*

Hence, storing the set of minimal support clauses (labels) does *not* improve the efficiency of label updating; what it does facilitate is $O(1)$ query processing of label sets for literals.

3.1.1 Interpretation Construction

The Interpretation Construction problem was defined as:

> Interpretation Construction (Optimisation)
> INSTANCE: A set of minimal supports $\Xi^*(X)$ for $\mathbf{x}$ with respect to a clause set X.
> OBJECTIVE: Derive a minimal expression $\mathcal{F}$ which computes $F = \bigwedge_i X_i$?

This is an optimisation problem, as we are finding an expression of minimal cost. Optimisation problems contain decision problems as sub-problems. Consider the Travelling Salesman Problem (TSP) [16]. In an instance of TSP the input is a set of n cities and the distances between every pair of cities. A tour is a closed path that visits every city exactly once. A decision problem for the TSP is to discover if there exists a tour of length $\leq k$ for some constant k. The decision problem has a *yes/no* answer. The optimisation problem is to find the tour of minimal length. Obviously, to find a tour of length k^*, the decision version of TSP must be solved for all values of $k \leq k^*$.

In formulating the problem solved by a monotonic JTMS [22], we see that it can be roughly stated as determining the satisfiability of a set X of clauses. This JTMS satisfiability task thus finds any solution, i.e. it solves the equivalent of the Decision problem. In contrast, ATMS Interpretation Construction (IC) finds the *minimal* expression (or *all maximal* solutions), which we will show to be harder than finding *any* solution (the decision problem).

We begin by looking at the decision problem, which we define as:

[13] A similar result has also been shown by Kean and Tsiknis [19].

Interpretation Construction Decision:
INSTANCE: The set $\Xi^*(X)$ of m minimal supports for a set $\mathbf{x}$ of literals with respect to clause set X, and an integer $k \leq m$.
QUESTION: Does there exist a Boolean expression F of cost $\leq k$?

ATMS Interpretation Construction Decision is now proven to be NP-complete by transforming an NP-complete problem, SET COVERING (cf. [16]) into the Interpretation Construction (IC) problem.

SET COVERING is defined as follows [16]:

Set Covering
INSTANCE: A family $Z = \{Z_1, ..., Z_m\}$ of subsets of a finite set of variables $V = \{V_1, ..., V_n\}$, and an integer $k \leq m$.
QUESTION: Is there a subfamily K of Z containing k sets such that K covers V, i.e. $\bigcup_{V_j \in K} V_j = V$?

The theorem derived is:

Theorem 4 (Interpretation Construction Decision) *ATMS Interpretation Construction Decision is NP-complete.*

Proof: First, it must be shown that Interpretation Construction (IC) (the decision version) is in NP. It is simple to design an algorithm which can guess an expression F' of cost $\leq k$ for F and can check in polynomial time whether F' computes F.

Second, a transformation from set covering to Interpretation Construction (IC) is outlined., A polynomial transformation $f(V, Z, k)$ from SET COVERING to IC is required, where SET COVERING is defined by V, k and Z, and IC defined by $(\mathbf{x}, \Xi^*, k)$. The transformation $f(V, Z, k) = (\mathbf{x}, \Xi^*, k)$ is as follows:

From each subset Z_j construct a prime implicate $\pi_j(X)$ such that $V_i \in Z_j$ iff $x_i \in \pi_j(X)$. By Lemma 4, if x is a literal, then there is a minimal support clause for x such that $\pi(x, X) = \{x\} \cup \xi(x, X)$. By this construction, for every $V_i \in Z_j$ there exists a minimal support set $\Xi_j^*(X)$ such that x_i is supported by $\xi_j^*(x_i, X) = \pi_j(x, X) - \{x\}$. In addition, we call the expression $F' = \bigwedge_i \pi_i$.

It is easy to see that $f(V, Z, k)$ constructs $(\mathbf{x}, \Xi^*, k)$ in polynomial time from V, k and Z.

Now, it is to be shown that $\exists$ a Boolean expression F' of cost $\leq k$ which computes F if and only if (V, Z, k) has a cover $K \subseteq Z$ such that $\mid K \mid = k \leq n$.

$\Longrightarrow$: Suppose $(\mathbf{x}, \Xi^*, k)$ has an expression F of cost $\leq k$, and every literal in F has support. Thus, there exists a set Π of k prime implicates such that every literal x_i occurring in F has support. Hence, every $V_j \in V$ is in the cover K by the construction. Thus, K is a set cover for V such that $\mid K \mid \leq n$.

$\Longleftarrow$: Suppose (V, Z, k) has a set cover $K \subseteq Z$ such that $\mid K \mid \leq n$. Then every $V_i \in V$ occurs in some subset Z_j. By our construction, this means that every every x_i occurring in F has support, in which case from $(\mathbf{x}, \Xi^*, k)$ $\exists$ a corresponding set of prime implicates Π' such that $\exists$ an expression F' of cost $\leq k$ and $F' = \bigwedge_{j:\pi_j \in \Pi'} \pi_j$. ■

However, the actual implementation of IC is even more complicated, as it discovers a *minimal expression* rather than any expression of cost $\leq k$.

We show that the complexity of the IC problem is given by Theorem 5.

Theorem 5 *ATMS Interpretation Construction is NP-hard.*

Proof: The decision version of IC is NP-complete. The full IC problem is no easier than the decision version of the problem. It cannot be proven that IC is in NP. We show this as follows. Assume that there exists a minimal expression with cost m^*. There is no shorter certificate for a yes instance verifying that this expression indeed has minimal cost other than a listing of all other Boolean expressions with cost $\geq m^*$. (The decision version of the problem can be used as a subroutine to test if the expression cost is $\leq m^*$.) The number of Boolean expressions can be as large as 2^{n-1}, so this certificate cannot be checked in polynomial time. Hence. IC is NP-hard. ■

4 RELATED WORK

To our knowledge, the only other studies of ATMS complexity are those of Dechter [13] and Brown et al. ([2], [3]). Dechter uses a Constraint Satisfaction formulation of the ATMS to examine the size of the label set. Dechter also analyses the size of this label set under various restrictions, such as when the constraint graph formed by the clauses input to the ATMS forms a tree.

Brown et al. formulate a Boolean-lattice-theoretic model of truth maintenance. In [2] they analyse the computational complexity of this abstract model. This model is more general, and because of this greater degree of generality is more computationally intractable than the one we propose. In contrast, we define the complexity of the actual problems the ATMS solves, and not of an abstraction of those problems.

There are several related problems, primarily in diagnostic reasoning, which are receiving considerable attention, and whose complexity has been defined. Closely related to the interpretation construction problem we investigate is the Generalized Set Covering approach used to solve Diagnostic Problems by Reggia et al. ([30], [31], [32]) and the Hypothesis Assembly approach of Allemang-et-al [1]. This set covering approach is essentially identical to the ATMS's approach. The ATMS's process of polynomial minimisation corresponds to Reggia's identification of the irredundant covers. Reggia's SET COVERING problem is finding a minimal cover for a collection of n subsets of a set of facts from a subcollection of $k \leq n$ of these subsets, and an irredundant covering is the set of minimal set covers. The NP-completeness of the set cover problem underlying this approach has been noted in both [31] and [1].

Within switching theory/Boolean algebra there are many versions of the ATMS prime implicate and interpretation construction algorithms. Examples include the Boolean minimisation methods of Quine and McCluskey [23], Tison [35] and Hwang et al. [18]. The complexity of such approaches has been studied in, for example, [36]. The main difference with these approaches is that they assume a fixed database, whereas the ATMS assumes labile

databases (due to exogenous, i.e. IE, input), and hence uses *incremental* prime implicate and minimal polynomial algorithms.

5 DISCUSSION

5.1 The Problem of Encoding in the ATMS

The parity problem described in Section 3.1 emphasises what we call the ATMS's "problem of encoding". By this we mean the difficulty of using the problem solver to find an encoding, or means of defining input clauses, assumptions and consumers, of the specific problem which ensures efficiency. For example, de Kleer in [9] provides a method of avoiding the combinatorial behaviour of the ATMS in solving the parity problem by generating solutions incrementally, i.e. reverting to JTMS-style problem solving. We note however that one big problem with this "solution" is that the ATMS generates a label set exponential in the number of database literals n. This solution stops the generation of an exponential number of solutions; i.e. incremental solution generation stops the interpretation construction process from generating all solutions simultaneously. There is no way to stop the generation of the full (exponential) label set.

Surprisingly, no mention is made about this problem of encoding in any literature on the ATMS, even though most ATMS users have to face this problem. We do not address this issue here, but raise it as an important topic in ATMS research, and use the parity problem as demonstration of how this encoding problem can occur. In brief, the moral is that one needs a good encoding to ensure efficiency of the ATMS.

This analysis also provides some intuition into how often a good encoding will be important in solving moderately large problems. de Kleer [9] divides the problem domain into three types of problems: problems with (1) many solutions, all of which are required; (2) one solution; and (3) many solutions, few of which are required. He claims that the ATMS has an advantage over a JTMS on class (1), an ATMS may be more efficient for class (2) dependent on the particular problem, and the standard ATMS approach must be reformulated for class (3).

We argue in [40] that there are few situations in which all solutions to a given problem are required, or can be efficiently computed. Hence, the main class of problems for which an ATMS is most efficient does not arise very often. Moreover, we argue that the encoding of the problem for both classes (2) and (3) is crucial for ATMS efficiency. An ATMS can simulate a JTMS, but unless a special encoding is used (such as the encoding for parity described in [9]) an ATMS will have a significantly larger overhead with respect to a JTMS. Moreover, this analysis has shown that almost all problems are as computationally difficult for the approach taken by an ATMS as parity, and almost all problems hence will require special encoding to ensure efficiency.

5.2 Complexity Results

The complexity results for the ATMS pertain to the problem of "compiling" the database X (as defined by Reiter and de Kleer [33]) into the support set for each database literal, and

performing Interpretation Construction to compute the minimal Boolean expression $\mathcal{F}$. All Boolean minimisation methods known to the author which can accept a general expression F compute $\Pi(X)$ first, and from $\Pi(X)$ compute $\mathcal{F}$. It has not been proven that Boolean minimisation entails computation of $\Pi(X)$, although no other general methods are known. Reiter and de Kleer [33] justify the computation of the support set by demonstrating the many uses of $\Xi^*(X)$. Any algorithm which computes $\Xi^*(X)$ must compute $\Pi(X)$ (by Lemma 4), and for such algorithms the complexity results derived here hold. The complexity results do not hold for algorithms which compute the minimal expression directly from F, and do not compute the set of minimal support.

These complexity results are quite negative and show the ATMS to be a computationally expensive general-purpose reasoning tool *for almost all problems.* However, such complexity results are not solely because of inherent design flaws in the ATMS, but also because intractability in the worst case is inevitable with any reasoning tool which determines all minimal supports for database literals.

This paper shows that the ATMS incurs significant computational overhead in providing the features which it does, such as constant-time query processing, almost no backtracking and simultaneous generation of all minimal interpretations. The first source of overhead, the average-case exponential time and space requirements associated with label (or minimal support set) generation, is unavoidable. Since the ATMS ensures generation of the complete label set, this label set will be of size exponential in either the number of literals (assumptions) or clauses in the database. This is a significant computational overhead to pay for constant-time query processing. Moreover, it raises the question of whether such a tradeoff is warranted, given that a JTMS will provide similar truth maintenance facilities at a fraction of the computational overhead. With a Horn-clause database over n literals and $O(n)$ clauses, a JTMS can identify a solution in linear time, answer queries restricted to a single context in constant time with $O(n^2)$ space requirements, and update its database in $O(n)$ time.

The second source of overhead, the simultaneous identification of all minimal solutions, is valuable for many problems, but is an NP-hard problem. Various control mechanisms may be employed to improve the efficiency of this stage of processing, e.g. sequential solution generation, as discussed above. However, they cannot circumvent the inherent intractability of the problem. Provan [26] has identified parameters which govern the search space expanded in interpretation construction, as well as a simple high-level visual recognition problem for which the interpretation construction process is exponential. This study indicates the existence of a range of problems for which the use of the ATMS to generate all minimal solutions is infeasible because of a combinatorial explosion during interpretation construction.

Given this intractability, one obvious question which arises is: Are there ways of designing TMSs to avoid intractability as much as possible? If the same functions are required (e.g. maintenance of the complete label set for all literals, simultaneous computation of all minimal solutions), such intractability is inherent in the problem. The ATMS already incorporates many methods (including heuristics) to improve efficiency. For example, the ATMS uses bit-vector representations to speed up the set operations it frequently performs. It also has several modes of operation, the default mode being the most efficient, but having the least deductive

power. For example, it can run in a mode which includes one-of-disjuncts or of arbitrary (i.e. not necessarily Horn) propositional clauses. More deductively powerful modes (such as the two just listed) can be invoked, but only as the situation demands, as they slow down the ATMS. de Kleer [8] describes these facilities in greater detail. The unavoidable fact is that the problems being solved are inherently exponential or NP-hard.

One alternative is to relax the type of solutions which are being searched for. If approximate solutions are adequate, the efficiency of the ATMS can be significantly improved. For example, Provan ([28], [27]) describes a method of assigning to ATMS assumptions weights so that only the "most likely" partial interpretations are explored, thus enabling the Problem Solver to reduce the search space through a form of best-first search.

Another alternative is to relax the level of dependency recording in the ATMS; namely, to maintain a restricted label set, thus avoiding the inherently exponential time and space requirements of the full label set. This, as well as other means of improving efficiency, are currently being studied.

Acknowledgements: Mike Brady, Johan de Kleer, Rina Dechter and Bill McColl all contributed useful comments.

References

[1] D. Allemang, M.C. Tanner, T. Bylander, and J. Josephson. Computational complexity of hypothesis assembly. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1112–1117, 1987.

[2] D. Benanev, A.L. Brown, and D.E. Gaucas. An Algebraic Foundation for Truth Maintenance. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 973–980, 1987.

[3] D. Benanev, A.L. Brown, and D.E. Gaucas. Reason maintenance from a lattice-theoretic point of view. Technical report, General Electric Co., 1987.

[4] J. Bowen and J. Mayhew. Consistency Maintenance in the REVgraph Environment. Technical Report AIVRU 020, University of Sheffield, 1986.

[5] A.K. Chandra and G. Markowsky. On the Number of Prime Implicants. *Discrete Mathematics*, 24:7–11, 1978.

[6] J. de Kleer. An Assumption-based TMS. *Artificial Intelligence Journal*, 28:127–162, 1986.

[7] J. de Kleer. Choices Without Backtracking. In *Proceedings of the American Association for Artificial Intelligence*, pages 79–85, 1984.

[8] J. de Kleer. Extending the ATMS. *Artificial Intelligence Journal*, 28:163–196, 1986.

[9] J. de Kleer. Problem Solving with the ATMS. *Artificial Intelligence Journal*, 28:197–224, 1986.

[10] J. de Kleer and J. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence Journal*, 24:7–83, 1980.

[11] J. de Kleer and G. Sussman. Propagation of Constraints Applied to Circuit Analysis. *Circuit Theory and Applications*, 8, 1980.

[12] J. de Kleer and B. Williams. Diagnosing Multiple Faults. *Artificial Intelligence Journal*, 32:97–130, 1987.

[13] R. Dechter. A Distributed Algorithm for the ATMS. Technical Report R-109, UCLA Department of Computer Science, 1988.

[14] W.F. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[15] J. Doyle. A Truth Maintenance System. *Artificial Intelligence Journal*, 12:231–272, 1979.

[16] M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, 1979.

[17] M. Herman and T. Kanade. Incremental Reconstruction of 3D Scenes from Multiple, Complex Images. *Artificial Intelligence Journal*, 30:289–341, 1986.

[18] H. Hwang and D. Chao. A New Technique for the Minimization of Switching Functions. In *Proc. IEEE Southeastcon 1985*, pages 299–304, 1985.

[19] A. Kean and G. Tsiknis. An Incremental Method for Generating Prime Implicants/Implicates. *Journal of Symbolic Logic*, to appear, 1989.

[20] V. Kuznetsov. On the Lower Estimate of the Length of the Shortest Disjunctive Normal Forms for almost all Boolean Functions. *Veroiatnostnye Metody v. Kibernetike*, 19:44–47, 1983.

[21] O.B. Lupanov. On the Realization of Functions of Logical Algebra by Formulae of Finite Classes (Formulae of Limited Depth) in the Basis $\cdot, +, -$. *Problemy Kibernetiki*, 6, 1965.

[22] D. McAllester. Reasoning Utility Package User's Manual. Technical Report AIM-667, MIT AI Laboratory, 1982.

[23] E.J. McCluskey. Minimization of Boolean Functions. *Bell System Technical J.*, 35:1417–1444, 1956.

[24] D. McDermott. Contexts and Data Dependencies: A Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):237–246, 1983.

[25] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, NJ, 1982.

[26] G. Provan. Complexity Analysis of Multiple-Context TMSs in Scene Representation. In *Proceedings of the American Association for Artificial Intelligence*, pages 173–177, 1987.

[27] G. Provan. Solving Diagnostic Problems Using Extended Truth Maintenance Systems. Technical Report 88-10, University of British Columbia, Department of Computer Science, 1988.

[28] G. Provan. Solving Diagnostic Problems Using Extended Truth Maintenance Systems. In *Proceedings of the European Conference on Artificial Intelligence*, pages 547–552, 1988.

[29] G. Provan. The Visual Constraint Recognition System (VICTORS): Exploring the Role of Reasoning in High Level Vision. In *Proc. IEEE Workshop on Computer Vision*, pages 170–175, 1987.

[30] J.A. Reggia, D.S. Nau, and P.Y. Wang. Diagnostic Expert Systems Based on a Set Covering Model. *Int. J. Man-Machine Studies*, 19:437–460, 1983.

[31] J.A. Reggia, D.S. Nau, and P.Y. Wang. A Formal Model of Diagnostic Inference. I. Problem Formulation and Decomposition. *Information Sciences*, 37:227–256, 1985.

[32] J.A. Reggia, D.S. Nau, and P.Y. Wang. A Formal Model of Diagnostic Inference. II. Algorithmic Solution and Application. *Information Sciences*, 37:257–285, 1985.

[33] R. Reiter and J. de Kleer. Foundations of Assumption-based Truth Maintenance Systems: Preliminary Report. In *Proceedings of the American Association for Artificial Intelligence*, pages 183–188, 1987.

[34] R. Stallman and G. Sussman. Forward Reasoning and Dependency Directed Backtracking in a System for Computer-aided Circuit Analysis. *Artificial Intelligence Journal*, 9:135–196, 1977.

[35] P. Tison. Generalization of Consensus Theory and Application to the Minimization of Boolean Functions. *IEEE Transactions on Electronic Computers*, EC-16(4):757–764, 1977.

[36] I. Wegener. *The Complexity of Boolean Functions.* John Wiley and Sons, 1987.

[37] B. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of the American Association for Artificial Intelligence*, pages 105–112, 1986.

[38] Y.I. Zhuravlev. Set-theoretical Methods in the Algebra of Logic. *Problemy Kibernetiki*, 8, 1982.

[39] D. McAllester. A Widely Used Truth Maintenance System. unpublished Technical Report, MIT AI Laboratory, 1985.

[40] G. Provan. An Analysis of Model Minimisation Methods of Computing AI Theories. Technical Report to appear, University of British Columbia, Department of Computer Science, 1989.