July 1992

Focusing ATMS Problem-Solving: A Formal Approach

Teow-Hin Ngair
University of Pennsylvania

Gregory Provan
University of Pennsylvania

Focusing ATMS Problem-Solving: A Formal Approach

Abstract

The Assumption-based Truth Maintenance System (ATMS) is a general and powerful problem-solving tool in AI. Unfortunately, its generality usually entails a high computational cost. In this paper, we study how a general notion of cost function can be incorporated into the design of an algorithm for focusing the ATMS, called BF-ATMS. The BF-ATMS algorithm explores a search space of size polynomial in the number of assumptions, even for problems which are proven to have exponential size labels. Experimental results indicate significant speedups over the standard ATMS for such problems. In addition to its improved efficiency, the BF-ATMS algorithm retains the multiple-context capability of an ATMS, and the important properties of consistency, minimality, soundness, as well as the property of bounded completeness. The usefulness of the new algorithm is demonstrated by its application to the task of consistency-based diagnosis, where dramatic efficiency improvements, with respect to the standard solution technique, are obtained.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-61.

Focusing ATMS Problem-Solving: A Formal Approach

MS-CIS-92-61
GRASP LAB 326

Teow-Hin Ngair
Gregory Provan

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

July 1992

FOCUSING ATMS PROBLEM-SOLVING: A FORMAL APPROACH

Teow-Hin Ngair *Gregory Provan*
Computer and Information Science Department
University of Pennsylvania
Philadelphia PA, 19104-6389, USA

Abstract

The Assumption-based Truth Maintenance System (ATMS) is a general and powerful problem-solving tool in AI. Unfortunately, its generality usually entails a high computational cost. In this paper, we study how a general notion of cost function can be incorporated into the design of an algorithm for focusing the ATMS, called BF-ATMS. The BF-ATMS algorithm explores a search space of size polynomial in the number of assumptions, even for problems which are proven to have exponential size labels. Experimental results indicate significant speedups over the standard ATMS for such problems. In addition to its improved efficiency, the BF-ATMS algorithm retains the multiple-context capability of an ATMS, and the important properties of consistency, minimality, soundness, as well as the property of bounded completeness.

The usefulness of the new algorithm is demonstrated by its application to the task of consistency-based diagnosis, where dramatic efficiency improvements, with respect to the standard solution technique, are obtained.

1 INTRODUCTION

The ATMS is a useful and powerful general problem-solving tool which has been widely used within AI. The range of applications include nonmonotonic reasoning [McCarthy, 1980; Reiter, 1980], qualitative physics [Forbus, 1990], visual interpretation [Bodington *et al.*, 1990; Provan, 1990b] and diagnosis [de Kleer *et al.*, 1990; de Kleer and Williams, 1987b; Reiter, 1987]. The ATMS allows search to be simultaneously conducted on multiple contexts, where each context (also known as environment) defines a set of assumptions about the problem domain. However, this flexibility comes at the cost of a high overhead due to storing many (and sometimes exponentially many) sets of such assumptions. Controlling the problem solver does not alleviate the poor performance on many tasks, such as diagnostic reasoning [Reiter, 1987], as it is the ATMS's generation of an enormous number of environments which is often the cause of inefficiency [Provan, 1990c]. Moreover, it is not always necessary to compute all environments for some domains, such as diagnosis. Various methods have been proposed to focus the ATMS on certain subsets of the search space (e.g. [Dressler and Farquar, 1990]), but none have been both general and effective.

The goal of this study is to analyze the efficiency of the ATMS in solving problems in several domains, and devise efficient algorithms for problems the standard ATMS does not solve efficiently. A cost-function-based algorithm has produced significant efficiency improvements in the ATMS, as has been demonstrated for the domain of diagnostic reasoning.

This paper provides a coherent semantics for both the ATMS label representation and the ATMS algorithms. In addition, the use of cost functions for focusing is directly integrated into the ATMS algorithm; the semantics of this extension of the ATMS are simply and precisely defined.

The cost-function approach, although not new to the AI literature, has not been rigorously applied to ATMS-based problem solving to date. We present a simple theoretical framework for a cost function ϱ (and associated order $\leq$) which preserves the partial ordering induced by set inclusion, namely for sets A and B, $A \subseteq B \Rightarrow \varrho(A) \leq \varrho(B)$. Such a cost function can handle any real-valued cost, e.g. probabilistic or integer-valued costs. Using an integer-valued cost function for a best-first ATMS algorithm called BF-ATMS has efficiently solved several problems which takes the standard ATMS hours of CPU-time. In addition to this efficiency, the BF-ATMS algorithm guarantees sound, consistent and minimal ATMS labels. Although completeness cannot be guaranteed, the labels exhibit a "bounded" completeness property, which means that they contain every environment (consistent or inconsistent) with cost lower or equal to a given bound. The BF-ATMS algorithm is most useful in problems where one has a large number of possible solutions but only a small number of them are desirable. Having a general theory of cost functions is important because many *ad hoc* choices of cost function will not improve efficiency or may have undesirable properties, e.g. lack of soundness or completeness. This general theory provides *a priori* information about the feasibility and properties of a wide variety of cost functions.[1]

Incorporation of traditional heuristic search techniques (such as A^*) into the ATMS

[1]Fine-tuning cost functions to optimize performance is not addressed here.

is non-trivial because of the multiple-context search of the ATMS. Most heuristic search techniques attempt to compute the single best solution, and keep a stack of next-best solutions if the current best partial solution becomes sub-optimal. Hence, for single-context searches, heuristic algorithms are easily defined. In contrast, the standard ATMS has no notion of best solution, only consistent or inconsistent solutions, and it computes all solutions simultaneously.[2] Given the enormous number of environments generated simultaneously by the ATMS, one needs to merge the ATMS's efficient (*i.e.* minimal) representation of labels and its ability to avoid recomputing labels with a means of generating only those environments required for the particular application. With a naive application of heuristic search, it is not obvious how the proliferation of environments typical in the ATMS can be curtailed without a deep understanding of the ATMS algorithm.

The BF-ATMS algorithm uses the *bounded* basic ATMS algorithm as an environment generator and a non-redundant cache for the environments generated. Hence, the improved efficiency of the BF-ATMS algorithm is largely due to the ability of the modified ATMS algorithm (while remaining correct) to ignore environments over the cost bound in *each* of its execution steps and hence, to prevent them from incurring heavy costs in all later computations. Also, the compactness of the label representation helps to eliminate many representational and computational redundancies (see [de Kleer, 1986a]) which is why the BF-ATMS algorithm is superior to a simple generate-and-test best-first search algorithm.

In the following, we formally define the basic ATMS algorithm and discuss its complexity and some pathological problems. Some existing focusing techniques for overcoming the complexity trap of an ATMS are described and we show that they are instances of a more general method of incorporating cost functions into the ATMS algorithm. We proceed to describe the BF-ATMS algorithm, and show how it can help solve the task of consistency-based diagnosis more efficiently. Finally, we compare our approach to existing work.

2 ATMS REVIEW

First, let us review a formal theory of the ATMS. What has been lacking in most previous formalizations, a semantics for ATMS operations like label computation, is described here in a uniform, implementation-independent framework.[3]

The following description and definitions of the ATMS are adapted from [Gunter *et al.*, 1990; Ngair, 1992]: An ATMS is specified by a set Q of propositional atoms, a set $\mathcal{F}$ of input formulas constructed from Q, a distinguished element $\bot \in Q$ denoting falsity and a set $\mathcal{A} \subseteq Q$ of propositional atoms, called the *assumptions.* Usually, one refers to a subset of $\mathcal{A}$ as an *environment* and the set of all environments as the environment lattice, *i.e.* the power set lattice for $\mathcal{A}$. In addition, some propositional atoms are identified as *premises*, which are

[2]Here we loosely use solution for context.

[3]As an example, the best-known semantics, that proposed in [Reiter and de Kleer, 1987], formalizes the labels, but not the label-computation operation. Similarly, the Boolean-lattice formalization of [Brown *et al.*, 1987] describes the ATMS as solving Boolean-lattice equations, but ignores the algorithms which are responsible for making the ATMS efficient.

considered to be always true. A propositional literal is either a propositional atom or the negation of a propositional atom. A clause is a finite disjunction of propositional literals, with no literal repeated. In the basic ATMS, only Horn clauses are allowed in $\mathcal{F}$, where a Horn clause is a clause with at most one positive literal. In the extended ATMS, DNF formulas of assumptions are also allowed. In this paper, we will only be concerned with the basic ATMS.

For each propositional atom X appearing in Q, an ATMS is required to compute the following set of environments [Reiter and de Kleer, 1987]:

$$\{p \subseteq \mathcal{A} \mid \bar{p} \vee X \text{ is a prime implicant of } \mathcal{F}\}, \tag{1}$$

where p is interpreted as a conjunction of literals and $\bar{p}$ is the negation of p. Because of this logical interpretation, it is more natural to order the environments by the superset ordering $\supseteq$.

An ATMS *label*, denoted by L_X, is a set of environments associated with each propositional atom X appearing in Q. An ATMS algorithm is supposed to manipulate the labels so that when it terminates, each label will have the correct values as stipulated by equation (1). In an ATMS, we are only interested in labels that are *anti-chains*, *i.e.* no environment is a subset of another environment in the same label. The following defines the operations that manipulate the labels in an ATMS:

Definition: Given two labels A and B, we say that A is *subsumed* by B, written as $A \preceq B$, if for every environment $p \in A$, there exists environment $p' \in B$ such that $p' \subseteq p$. The operations *meet*, *join* and *difference* of A and B are defined as:

$$\begin{aligned} A \wedge B &= \mathcal{MAX}(\{p_1 \cup p_2 \mid p_1 \in A, p_2 \in B\}); \\ A \vee B &= \mathcal{MAX}(A \cup B); \\ A - B &= \{p \in A \mid \nexists p' \in B \text{ s.t. } p' \subseteq p\}, \end{aligned}$$

where $\mathcal{MAX}$ is the function which returns the maximal environments (smallest subsets) under the superset ordering $\supseteq$. □

It is easier to characterize the above operations by referring to the set of environments *supported* by the labels:

Definition: The *support* of a label A is defined to be the set of environments that are supersets of at least an environment in A, *i.e.*

$$\mathcal{U}(A) = \{p \subseteq \mathcal{A} \mid \exists p' \in A \text{ s.t. } p' \subseteq p\}$$

Note that $\mathcal{MAX}(\mathcal{U}(A)) = A$ because of the anti-chain property of a label. The following results summerize the important properties of the operations defined on labels:

Proposition 1 *The following properties hold for the meet, join and difference operations on any pair of labels A, B:*

1. $\mathcal{U}(A \wedge B) = \mathcal{U}(A) \cap \mathcal{U}(B)$.

2. $\mathcal{U}(A \vee B) = \mathcal{U}(A) \cup \mathcal{U}(B)$.

3. $\mathcal{U}(A - B) = \mathcal{U}(\mathcal{U}(A) - \mathcal{U}(B))$. $\square$

Proof: Immediate. $\square$

The first task an ATMS algorithm needs to perform is to initialize the labels.

Definition: Given an ATMS with a set of input formulae $\mathcal{F}$, different possible values of L_X's (where $X \in Q$) are called *states* of the ATMS. An *initial ATMS state* is a state such that for each $X \in Q$, the following is true:

1. If X is an assumption but not a premise, then L_X is equal to $\{\{X\}\}$,
2. if X is a premise, then L_X is equal to $\{\{\}\}$. The corresponding justification with empty antecedent can be deleted from $\mathcal{F}$,
3. otherwise, L_X is equal to the empty set. $\square$

Semantics: The semantics of the ATMS can be characterized by specifying that the output of the ATMS, the L_X's, must satisfy

$$\begin{aligned} L_\perp &= \mathcal{MAX}(\{p \subseteq \mathcal{A} \mid \mathcal{F} \cup p \text{ is inconsistent}\}), \\ L_X &= \mathcal{MAX}(\{p \subseteq \mathcal{A} \mid \mathcal{F} \cup p \text{ is consistent, and } \mathcal{F} \cup p \models X\}), \end{aligned}$$

where $X \not\equiv \perp$, and $\mathcal{F} \cup p$ denotes $\mathcal{F} \cup \{\Rightarrow x \mid x \in p\}$.

Algorithm: The next definition characterizes the fundamental step in a basic ATMS algorithm.

Definition: For any justification $\psi \, (\equiv X_1, \ldots, X_n \Rightarrow Y) \in \mathcal{F}$, we say that ψ is *applicable* in the ATMS if in the current state of the ATMS, we have

$$L_{X_1} \wedge \cdots \wedge L_{X_n} \not\preceq L_Y. \qquad (2)$$

Furthermore, the *application* of an applicable ψ is the modification of L_Y as follow:

$$L_Y \vee (L_{X_1} \wedge \cdots \wedge L_{X_n}), \qquad (3)$$

and if $Y \equiv \perp$, we perform the additional step of removing the new inconsistent environments from every $X \not\equiv \perp$:

$$L_X \;\leftarrow\; L_X - L_\perp. \quad \square$$

Intuitively, a justification is said to be applicable if some environments are already known to entail every antecedent node but are not yet included in the label of the consequence node. The application of the justification is then a rectification of this problem. Furthermore, if the consequence node represents the inconsistency, then the newly added inconsistent environments are removed from every other node so as to reduce duplicity. These definitions can be extended to a sequence of justifications as follows:

Definition: Let $(\psi_1, \ldots, \psi_s)$ be a sequence of justifications. The sequence is said to be *applicable* to the ATMS if ψ_1 is applicable and each $\psi_i, 1 < i \leq s$ is applicable after the sequential application of $\psi_1, \ldots, \psi_{i-1}$ to the ATMS. The *application* of the sequence to the ATMS is defined to be the sequential application of $\psi_1, \ldots, \psi_s$. An applicable sequence of justifications is said to be *complete* with respect to $\mathcal{F}$ if, after the application of the sequence, no justification in $\mathcal{F}$ is applicable. $\square$

A *basic ATMS algorithm* is a process which repeatedly finds an applicable justification and applies it, *i.e.* searching for a complete applicable sequence and applies it. This "batch" algorithm can also be used to update the ATMS "incrementally". When new justifications are added to the ATMS, one can simply execute the algorithm starting with the ATMS state resulted from the previous run. This gives the same result as running the ATMS algorithm starting with the initial ATMS state again. The above algorithm has been shown ([Gunter *et al.*, 1990]) to converge with the results as specified in equation (1).

3 COMPLEXITY RESULTS

It is known that the task solved by the ATMS is hard [Provan, 1990c]. In particular, note that computing the label for even a single literal will be NP-hard. As a worst case example, consider an ATMS with the following input set of clauses:

$$\begin{gathered} A_i \Rightarrow B_i, \quad i \in [1, n] \\ B_1, \ldots, B_n \Rightarrow z. \end{gathered}$$

It is easy to check that the label of z is $\{\{X_1, X_2 \ldots X_n\} \mid X_i \equiv A_i \text{ or } X_i \equiv B_i, 1 \leq i \leq n\}$. Therefore, an ATMS will generate a 2^n size label for the node z. Hence, we have:

Proposition 2 *A basic ATMS with $n+1$ justifications and $2n$ assumptions may generate label with size as large as 2^n.* $\square$

The above result establishes an exponential lower bound for the complexity of any ATMS algorithm that is complete. Similar results were obtained for the case of prime implicates/implicants generation in [Chandra and Markowsky, 1978]. Furthermore, we also know that implementing the ATMS in terms of a prime implicate generation algorithm [Kean and Tsiknis, 1990; Reiter and de Kleer, 1987] will lead to average-case performance exponential in the number of literals or input formulae [Provan, 1990c].

For general cases, the NP-hardness of the ATMS task does not tell us exactly what factors affect the complexity. The following result remedies this by giving an upper bound to the complexity of the basic ATMS algorithm:

Proposition 3 *Given a knowledge base with c assumptions and formulae with n distinct propositional atoms and l total occurrences of propositional atoms, a worst case complexity of running the basic ATMS algorithm is $O(c * l * n * 2^{3c})$.*

Proof: First, we consider the meet and join operations as defined in formulas (2) and (3). Given two labels C_1 and C_2 of sizes m_1 and m_2 respectively, the worst case complexities of computing $C_1 \wedge C_2$ and $C_1 \vee C_2$ are $O(c * (m_1 * m_2)^2)$ and $O(c * (m_1 + m_2)^2)$ respectively. Note that $O(c)$ is the cost for each subset comparison.

Since the size of the environment lattice P is 2^c, every L_X of the ATMS has at most $O(2^c)$ environments.[4] This is also true for the results of the meet and join of two L_X's, hence the worst case complexities of the operations meet and join are both $O(c * 2^{2c})$.[5] Therefore, the application of a justification of length k, *e.g.* $x_1, \ldots, x_k \Rightarrow y$, requires at most the cost $k * O(c * 2^{2c})$.

To include the cost of finding a complete applicable sequence, note that one may need to search through most of the justifications, *i.e.* need $O(l)$ meet and join operations, before finding one that is applicable. Furthermore, due to the monotonicity of application, any complete applicable sequence of justifications is of length at most $n * 2^c$. Therefore, a worst case complexity of running the basic ATMS algorithm is $O(c * l * n * 2^{3c})$. □

An interesting special case is if there is no assumption given, then the complexity of the ATMS algorithm reduces to $O(l * n)$. In particular, such situations arise when we wish to check the satisfiability of a collection of propositional horn clauses. For instance, given an ATMS, one might wish to check whether a particular subset of assumptions is consistent with the input formulae. This can be accomplished by changing these assumptions to premises and removing all assumptions from the system before running the ATMS algorithm. We will know that the given assumptions are consistent if the label of $\perp$ is empty when the algorithm terminates. It is worth noting that the complexity of checking satisfiability using an ATMS compares favoritely with other existing algorithms [Chang and Lee, 1973; Dowling and Gallier, 1984].

Since the incremental basic ATMS algorithm is an extension of the batch ATMS algorithm, it also has $O(c * l * n * 2^{3c})$ as an upper bound to its worst case complexity. In the following section, special cases of tasks will be analyzed to identify some pathological cases where these exponential complexity results hold.

[4]Note that we are oversimplifying here. The largest possible size of a label is $C^c_{c/2}$. Using Sterling's formula, this reduces to $O(2^c/\sqrt{c})$.

[5]This requires suitable data structures for storing pairwise unions of environments during the meet operation, *e.g.* a 2^c size array, so that each environment needs to compare with not more that 2^c environments.

4 DIFFICULT CASES FOR THE ATMS

This section characterizes a class of problems for which the ATMS will produce labels of size exponential in the number of assumptions. These problems illustrate the different ways in which the multiplying effect of combining labels from the antecedents of a justification can lead to an explosion in the size of the consequent-node's label. By learning these pathological cases, we hope to devise ways of controlling the multiplying effect so as to reduce the complexity of the ATMS computation. A step towards this goal is described in Section 7.

In the following, upper-case letters (e.g. A, B) refer to assumptions.

Parity Problem

The parity problem is one for which the complexity of even the basic ATMS is provably exponential [Provan, 1990b]. The parity problem is defined (as given in [de Kleer, 1986c]):

For an expression F of n variables $\{p_1, \ldots, p_n\}$, 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 x_i the parity for all variables up to and including variable p_i. Define assumptions A_i as $p_i = 0$ and B_i as $p_i = 1$ for $i = 1, ..., n$, and assumptions X_i as $x_i = 1$ and Y_i as $x_i = 0$ for $i = 1, ..., n$. Hence, one obtains the boundary condition (*i.e.* premise) x_0, and for $i = 1, ...n$,

$$\begin{aligned} x_{i-1}, A_i &\Rightarrow y_i, \\ y_{i-1}, B_i &\Rightarrow y_i, \\ x_{i-1}, B_i &\Rightarrow x_i, \\ y_{i-1}, A_i &\Rightarrow x_i. \end{aligned}$$

This gives a total of $4n + 1$ input Horn clauses based on n variables indicating whether the i^{th} bit is 1 (A_i) or 0 (B_i), in addition to $n + 1$ added variables indicating whether the i^{th} prefix sub-string is of odd (y_i) or even (x_i) parity. Given this set of input clauses, it has been shown that:

Proposition 4 *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 prime implicates of length n each.*

Proposition 4 was first proven by Lupanov [Lupanov, 1965]. It has been cited in the AI literature by McAllester [McAllester, 1985], and discussed by de Kleer [de Kleer, 1986c]. This proposition shows yet another expression F with size $O(n)$ such that any minimal expression $\mathcal{F}$ which computes F must have $O(2^n)$ size. An immediate corollary is:

Corollary 5 *Enumerating the set of minimal support clauses for the parity problem is of complexity exponential in the number n of components.* □

A solution to this exponential blowup, generating the labels for the variables one-by-one on demand, has been proposed in [de Kleer, 1986c]; this solution does have nice complexity properties, even though it is *ad hoc* and does not generalize to other problems.

Diagnostic Reasoning

Diagnostic reasoning from first principles has been an area of intensive research [de Kleer and Williams, 1987b; Reiter, 1987] and many of the existing diagnostic systems employ an ATMS in actual implementations [de Kleer and Williams, 1987b; de Kleer and Williams, 1989]. A recent formal study of diagnostic reasoning [de Kleer *et al.*, 1990] has characterized the diagnosis problem as computing the prime implicants of the system description and set of observations. This characterization of the diagnosis task in terms of prime implicates/implicants makes it not a surprise that the ATMS solving this task is usually of exponential complexity. For instance, consider a circuit inspired by the pathological example used in the proof of Proposition 2. This is shown in figure 1.

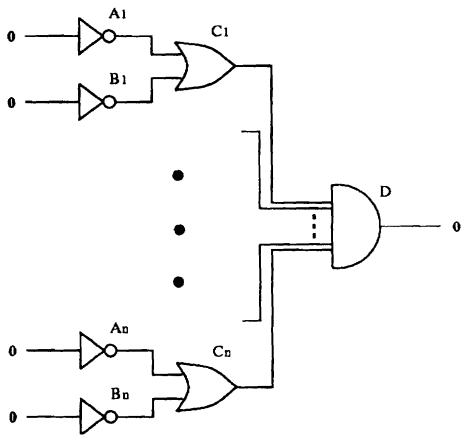

Figure 1: A circuit with an exponential number of conflicts

In this problem, the output of the circuit (0) does not correspond to the expected output (1). One can deduce from this observation that the collection of conflict sets is $\{\{\text{AB}(X_1), \ldots, \text{AB}(X_n), \text{AB}(C_1), \ldots, \text{AB}(C_n), \text{AB}(D)\} \mid X_i \equiv A_i \text{ or } X_i \equiv B_i\}$ which is exponential in size. However, the set of kernel diagnoses is $\{\{\text{AB}(A_i), \text{AB}(B_i)\} \mid 1 \leq i \leq n\} \cup \{\{\text{AB}(C_i)\} \mid 1 \leq i \leq n\} \cup \{\text{AB}(D)\}$ which is linear in size. Therefore, to generate all minimal/kernel diagnoses by calculating the hitting sets of conflict sets [de Kleer *et al.*, 1990] is bound to be inefficient. We will propose a solution to this problem in a later section.

Nonmonotonic Encodings

In [Dressler, 1990], the following set of justifications are added to an ATMS to help calculate the coherent extensions of non-monotonic reasoning problems:

$$A_i \Rightarrow q_i, B_i \Rightarrow q_i, \qquad i \in [1, n]$$
$$q_1, \ldots, q_n \Rightarrow q$$

where each A_i represents an assumption and B_i represents the corresponding out-assumption, usually referred to as $out(A_i)$.

Observe that the above clauses are merely a restatement of the pathological case we used to prove Proposition 2. In particular, the former is equivalent to the transformation of the latter by eliminating the need of having justified assumptions, *i.e.* assumptions which are also consequences of some justifications. This is usually accomplished by having all justified assumptions replaced by new non-assumptions and adding new justifications which justify the new non-assumptions by the assumptions that they replace. In this case, we replace every B_i by a distinct non-assumption q_i and add the new justification $B_i \Rightarrow q_i$ to the set of clauses.

This observation tells us that the price of adding the above clauses to simulate non-monotonic reasoning in an ATMS is inherently expensive. In particular, in the absence of other justifications, the label of q will be $\{\{X_1, \ldots, X_n\} \mid X_i \equiv A_i \text{ or } X_i \equiv B_i\}$, which is of size 2^n.

5 IMPOSING BOUNDS ON THE ATMS

We now consider restrictions intended to curtail the worst case behavior of an ATMS. In particular, we will describe several global bounds which one can impose on the problem so that the ATMS algorithm will terminate in polynomial time.

Given a justification $X_1, \ldots, X_k \Rightarrow X$, the environments contributed by this justification to the node X can be as large as $N_{X_1} \times \cdots \times N_{X_k}$ where $N_{X_i}, 1 \leq i \leq k$, is the label size of X_i. Therefore, it is undesirable to have long justifications in general. The "obvious" fixes turn out to be ineffective. In particular, even if we restrict all the justifications to be short, *e.g.* of maximum length 3, we can still get exponential blow up in the size of a label due to some long chains of justifications.

Proposition 6 *Imposing a constant bound on the length of justifications does not solve the ATMS label explosion problem.*

Proof: Consider the following input formulas to an ATMS:

$$\begin{array}{rl} A_i \Rightarrow B_i & i \in [1, n] \\ b^j_{2*k-1}, b^j_{2*k} \Rightarrow b^{j+1}_k & k \in [1, n/2^j], j \in [1, \log n] \end{array} \tag{4}$$

where $b^1_i \equiv B_i$ and all other b's are non-assumptions. Note that the number of justifications is $2 * n - 1 = O(n)$. However, it is not hard to see that the label of $b^{\log n}_1$ is

$$\{\{X_1, \ldots, X_n\} \mid X_i \equiv A_i \text{ or } X_i \equiv B_i, 1 \leq i \leq n\}$$

which is $O(2^n)$ in size. Therefore, having a constant bound on the length of justifications does not solve the label explosion problem. □

Moreover, imposing a constant bound on the length of a chain of justifications does not help either:

Proposition 7 *Imposing a constant bound on the length of the longest chain of justifications does not solve the ATMS label explosion problem.*

Proof: Consider the example we used to prove Proposition 2 where the longest chain of justifications is only 2:

$$\begin{aligned} A_i &\Rightarrow B_i \qquad i \in [1,n] \\ B_i, \ldots, B_n &\Rightarrow z \end{aligned}$$

We know that the label of z is

$$\{\{X_1, \ldots, X_n\} \mid X_i \equiv A_i \text{ or } X_i \equiv B_i, 1 \leq i \leq n\}$$

Therefore, the label of z is of size 2^n for an input formula of size $n+1$. Thus, having a constant bound on the length of a justification chain does not curtail the exponential blow up in label size. $\square$

If we impose a constant bound on both the length of justifications and the length of chains of justifications, then we can indeed ensure that the label size will not blow up exponentially. This is because such cases can only arise when the size of assumption-support for each proposition is also bounded by a constant. Hence, there can be at most a (possibly large) constant number of environments in each label. Unfortunately, not many interesting problems can be encoded with such constraints. In the following, we describe other more successful attempts.

Small Assumption Sets

The complexity analysis of the basic ATMS algorithm shows that its worst case complexity grows polynomially with respect to the number of nodes and justifications, and exponentially only with respect to the number of assumptions. Therefore, if we can control the number of assumptions in a problem, then we can manage to get away from the exponential blow up problem. An obvious situation is when the number of assumptions is always bounded by a small constant. In this case, the complexity of the ATMS algorithms are simply quadratic. Even if the number of assumptions grows logarithmically with respect to the number of justifications or the number of nodes, the ATMS algorithm will still have a worst case complexity that is polynomial with respect to the number of justifications or the number of nodes.

Bounded Environment Length

If one knows more about the problem domains that one is dealing with, then one may be able to predict if the ATMS algorithms are going to blow up exponentially or not. There are also those cases where one is only interested in some particular subclasses of all possible solutions computable by the ATMS or interested in only approximate solutions. A particular case is when one is only interested in bounded length environments. By considering all the environments with a constant bound in their length, one is interested in having only *short explanations* for nodes in the ATMS. One can easily modify the ATMS algorithms so that they will only generate environments with length shorter than the bound (see Section 7 below) and hence make the ATMS algorithms more efficient.

It is easy to see that an ATMS with a bound on environment length can be incomplete. As a simple example, consider the parity problem, where each literal has the same label length α. The total size of the label is 2^n where n is the number of assumptions. Enforcing a small label, say $k < n$, can ensure every label is of manageable size, but will fail to produce any solutions. In an analogous manner, any restriction on the number of assumptions may prevent certain solutions (and possibly all solutions) from being discovered.

However, there is no underlying theory for most focusing strategies [Dressler and Farquar, 1990; Forbus and de Kleer, 1988; Ng and Mooney, 1991]. In the following, we show that the efficient examples just discussed, bounded assumption sets and environment lengths, are not merely *ad hoc*, but are examples of a broader theory. We introduce the notion of a *cost function* which subsumes these focusing techniques. This integration of cost functions into the ATMS can be done in a semantically consistent manner, given the semantics described in section 2.

6 COST FUNCTIONS

Polynomial-time behavior in the ATMS can be achieved by enforcing some form of cost function on the ATMS. The general theory of a cost function is first described, followed by some examples of different types of cost function.

Definition: A *cost function* $\varrho : \mathcal{A}' \rightarrow \mathbb{R}$ assigns a real-valued cost to any set $\mathcal{A}'$ of assumptions. ϱ induces a total ordering $\leq$ onto assumption sets. A cost function ϱ is said to satisfy the *monotonicity criterion* if $A \subseteq B \Rightarrow \varrho(A) \leq \varrho(B)$.

ATMS labels naturally observe the monotonicity criterion, since label minimality is determined by set inclusion. Assigning a partial ordering on assumption sets allows search to be focused on the least-cost environments. In any such search, it is desirable to maintain the important ATMS label properties, which were defined in [de Kleer, 1986a]: For every propositional atom X the ATMS is said to have the property

Soundness: if for all $p \in L_X$, $\mathcal{F} \cup p \models X$;

Consistency: if every $p \in L_X$ is consistent, *i.e.* $\mathcal{F} \cup p \not\models \perp$;

Minimality: if for all $p \in L_X$, there exists no $p' \subseteq \mathcal{A}$ such that $p' \subseteq p$ and $\mathcal{F} \cup p' \models X$;

Completeness: if for all p such that $\mathcal{F} \cup p \models X$, there exists $p' \in L_X$ such that $p' \subseteq p$.

Of these, it is most important to maintain soundness, consistency and minimality. If completeness is sacrificed, "approximate" solutions can be computed; even so, it is desirable to maintain admissibility (an optimal solution will be found if it exists). One such approximation is to maintain a "bounded" completeness in the solutions computed, in which environments with costs lower than a given bound are guaranteed to be generated. The BF-ATMS algorithm described in the next section is an efficient method for computing such approximate solutions.

Two broad classes of cost function, probabilistic and integer-valued, can be defined. A probabilistic cost function takes [0,1] values, and an integer-valued function takes value in $\mathbb{N}$. Both cases have advantages and disadvantages, some of which are summarized in Table 1.

Cost type	Advantages	Disadvantages
Probability	Exact—completeness preserved	Inefficient; Independence assumptions
Integer	Efficient	Heuristic—may be incomplete

Table 1: Advantages and disadvantages of probabilistic and integer-valued cost functions

These two classes of cost function are briefly examined, and their application to the ATMS is discussed.

Probabilistic Cost Functions

Consider the case of assigning a probability (i.e. [0,1] weight) to each assumption. A formal uncertainty calculus, Dempster-Shafer Theory, is obtained [Laskey and Lehner, 1990; Provan, 1989] by imposing certain restrictions on the problem description, such as assumptions being mutually exclusive and exhaustive, and computing the weights of labels exactly without pruning environments.

For this calculus, the cost of the label for literal x corresponds to the Dempster-Shafer Belief assigned to x, $Bel(x)$. The benefit of this approach is the precise semantics associated with Belief assignments. However, the drawback is that the #P-hardness of calculating $Bel(x)$ [Orponen, 1990; Provan, 1990a]. Hence, simulation, e.g. Monte-Carlo simulation [Wilson, to appear 1992], must be used to efficiently compute either of these tasks. However, this probabilistic approach has several drawbacks. Unless simulation is used, the computational savings of cost-based focusing may be lost due to the computational demands of computing the costs. In addition, the independence assumptions required are stringent, and may be unreasonable for many domains.

Heuristic methods of assigning weights to labels are an alternative means of efficiently computing a probabilistic partial ordering on labels. One suggested heuristic [de Kleer and Williams, 1987b] is to make the following assignment: for assumption set $\mathcal{A}'$, set $\varrho(\mathcal{A}') = 1 - \prod_{c \in \mathcal{A}'} p(c)$, where $p(c)$ is the probability of c, $0 \leq p(c) \leq 1$. In addition, one must assume that the c_i's are all mutually independent. The goal of this approach is to minimize $\varrho(\mathcal{A}')$.

Note, however, that this heuristic is just the dual to maximizing $\varrho(\mathcal{A}') = \prod_{c \in \mathcal{A}'} p(c)$. In other words, one is approximating $Bel(x)$ using $\max_{\mathcal{A}_i \in L_x} \varrho(\mathcal{A}_i)$, where $\varrho(\mathcal{A}_i) = \prod_{c \in \mathcal{A}_i} p(c)$.

This cost function, however, may not help improve problem-solving efficiency if the independent assumption fails, *e.g.* if we are only interested in environments resulted from the cross product of some assumption sets [de Kleer and Williams, 1987a].

Integer-Valued Cost Functions

Another heuristic approach is to assign integer-valued weights to assumption sets, *i.e.* $\varrho : \mathcal{A} \to \mathbb{N}$. The advantages of this approach over the case of probabilistic weights include avoiding required independence assumptions, or weight normalization. The disadvantage of not normalizing the weights is that only the induced total ordering of label costs is meaningful, and the relative magnitudes of the costs of the labels are not necessarily meaningful. In some cases, this is greatly outweighed by the efficient computation of partial orderings possible with this approach.

Simple cost functions, such as $\varrho(\mathcal{A}) = \sum_{A_i \in \mathcal{A}} \varrho(A_i)$, can be very effective. A particular instance of this cost function is $\varrho(A_i) = 1$ for all i, *i.e.* the cost is the length of an environment. In this case, the BF-ATMS algorithm described in the next section will find the minimum-length environments first. Furthermore, the time required to find a solution of length l is polynomial in n if l is constant.

The next section discusses the incorporation of the ATMS with cost functions in which the monotonicity criterion is satisfied, and the application of integer-valued cost functions to examples.

7 ATMS FOCUSED SEARCH ALGORITHM

Consider each application of a justification $(X_1, \ldots, X_n \Rightarrow Y)$ in the basic ATMS algorithm. First, we need to check the following condition:

$$L_{X_1} \wedge \cdots \wedge L_{X_n} \not\leq L_Y \qquad (5)$$

where

$$L_X \wedge L_Y = \mathcal{MAX}(\{p_x \cup p_y \mid p_x \in L_X, p_y \in L_Y\}), \qquad (6)$$

and $\mathcal{MAX}$ performs the subsumption checking and returns the maximal environments (minimal subsets) in the set. If equation (5) is found to be true, the ATMS algorithm proceeds to update the label of Y to $L_Y \vee (L_{X_1} \wedge \cdots \wedge L_{X_n})$ [and removing L_Y from every other L_X if $Y \equiv \perp$], where $L_X \vee L_Y = \mathcal{MAX}(L_X \cup L_Y)$. In each of the $\wedge$ operations, an environment in the first set is unioned with an environment in the second set as in equation 6, to produce a new environment. Hence, by the monotonicity criterion of a cost function and the abstract definition of the basic ATMS algorithm, it is clear that:

Lemma 8 *In each step of the basic ATMS algorithm, the cost of every newly generated environment is always larger than or equal to the costs of the environments which generate it.*

The above lemma is of great significance if we wish to set a cost upper-bound on the environments generated by the ATMS. It tells us that we will never block the generation of a desirable environment due to the elimination of environments which exceed the upper bound. Therefore, we may immediately disregard, during each step of the basic ATMS algorithm, many of the environments that would otherwise be generated by the standard ATMS algorithm and incur heavy (probably exponential) computational cost later. Consequently, the ***bounded basic ATMS*** algorithm defined below is guaranteed to generate all and only the correct environments with costs lower or equal to a given cost bound, and at the same time, achieve greatly reduced run time.

Definition: A *Bounded Basic ATMS* algorithm is the basic ATMS algorithm modified to accept a cost bound $\mathcal{B}$, a cost function ϱ, and with the $\wedge$ operation changed to:

$$L_X \wedge L_Y = \mathcal{MAX}(\{p_x \cup p_y \mid p_x \in L_X, p_y \in L_Y \text{ and } \boldsymbol{\varrho}(\boldsymbol{p}_x \cup \boldsymbol{p}_y) \leq \boldsymbol{\mathcal{B}}\}). \qquad (7)$$

Note that Lemma 8 is not generally true for other similar algorithms. In particular, it is neither true for the extended ATMS algorithm [de Kleer, 1986b] nor the consensus-based CMS algorithm [Kean and Tsiknis, 1990]. Therefore, if we wish to retain the completeness property, it is not possible to incrementally eliminate environments which exceed a bound in the extended ATMS or CMS algorithms; thus, a cost bound is not helpful in expediting solution computation.

In the following, we outline a best-first like search strategy for finding the minimum cost environments for any node Y in a basic ATMS using the bounded basic ATMS algorithm.

Procedure BF-ATMS (Y)

1. set the bound to be the lowest possible, *i.e.* the empty environment cost;
2. introduce all assumptions with cost lower or equal to the current bound;
3. run the bounded basic ATMS algorithm with the current cost bound;
4. if an environment appears in the label of Y, stop and report the result;
5. increase the bound to the next higher cost, goto step 2.

The above algorithm is a simple application of the heuristic search method using the bounded basic ATMS algorithm as an environment generator and a non-redundant cache for the environments generated. However, note that a direct incorporation of heuristic search methods (such as A^*) into the standard ATMS is not possible because of the enormous number of environments generated simultaneously, and it is not obvious how this proliferation of environments can be curtailed without a deep understanding of the algorithm. This difficulty has lead to the proposal of various much more specific focusing methods in the past [de Kleer and Williams, 1987a; Forbus and de Kleer, 1988; Dressler and Farquar, 1990].

The most significant feature of the BF-ATMS algorithm is that it generates each valid environment in a best-first fashion, *i.e.* it will generate all correct environments with lower costs before it generates the environments with next higher cost. In particular, the environments with minimum cost will always be generated first. Note that the environments generated during each run of the bounded basic ATMS algorithm are cached and need not be generated again when the cost bound is increased.

Again, we would like to point out that the improved efficiency of the BF-ATMS algorithm is largely due to the ability of the bounded basic ATMS algorithm (while remaining correct) to ignore environments over the cost bound in *each* of its execution steps and hence, to prevent them from incurring heavy costs in all later computation. Also, the compactness of the label representation helps to eliminate many representational and computational redundancies (see [de Kleer, 1986a]) which is why the BF-ATMS algorithm is superior to a simple generate-and-test best-first search algorithm.

In terms of the four desirable properties defined in Section 6, a characterization of the BF-ATMS algorithm is the following:

Proposition 9 *At the termination of each loop in the* BF-ATMS *algorithm, the labels generated will satisfy the sound, consistent and minimal properties of an ATMS. Furthermore, the labels also enjoy the "bounded" completeness property which means that they contain every environment (consistent or inconsistent) with cost lower or equal to a given bound.*

Proof: The modification to the meet operation as specified in equation (7) generates only a subset of what the normal meet operation generates. Hence, the soundness property still holds.

The modified meet operation also ensures that no environment of cost greater than the current bound will be generated. Furthermore, as a direct consequence of Lemma 8, the algorithm does not prevent environment of cost less than or equal to the current bound from being generated. Hence, consistency, minimality and bounded completeness follow. □

In some applications, we may want to find more than one solution. This can be achieved by changing the termination condition to:

4′. if enough environments for Y are found, stop and report the result.

Other similar variations can also be made to the algorithm. For instance, if we have a better understanding of the possible solution cost, we can set the initial bound of the algorithm closer to the solution instead of starting from scratch. Furthermore, with a better understanding of the solution space, it is also possible to incorporate heuristics which control the cost increment so as to obtain a faster convergence in running the algorithm.

One observation about the algorithm is that after the initial loop of the algorithm, one can check if the empty set $\{\}$ is in the label of $\perp$. If so, we can conclude immediately that the system is inconsistent and abort.

We shall use the example shown in equation (4) with $n = 16$ to illustrate some of the complexity issues regarding the BF-ATMS algorithm. Initially, we used the number of assumptions in an environment as its cost and obtained the empirical results shown in Table 2.

[6]The system ran out of memory after about 100 minutes of CPU-time.

Cost bound	Total labels size	Time (seconds)
0	0	0.00
1	48	0.02
2	80	0.08
4	144	0.26
8	656	15.0
16	66192	> one hour[6]

Table 2: Example run of BF-ATMS with various bounds

One can observe that the BF-ATMS algorithm has enabled us to generate the ATMS labels in an incremental fashion and the algorithm terminates rather rapidly for a low cost bound. However, the simple cost function is not very helpful in efficiently finding the minimum cost environments for all the nodes in our example, because the cost of every environment in the node b_1^5 is 16. But we can solve this problem by changing the cost function to indicate the preference of choosing assumptions of type A instead of type B. In particular, we can assign to each assumption the cost of 1 if it is of the form A_i, otherwise, the cost of 100. The sum of assumption costs is then used as the cost of an environment. With this modified cost function, the BF-ATMS can compute a minimum cost environment for every node in less than 0.2 second. With exactly the same cost function, albeit not a very efficient one, we can also generate an even parity and several odd parity 20-bit strings in about 0.5 second and 3 seconds respectively. In conclusion, the selection of a good (but not necessary the best) cost function for the given problem is essential for deriving the desirable solutions in a reasonable amount of time.

The question is: what type of cost function would induce the assignment of an ordering over the environments to guarantee polynomial performance in the ATMS? To ensure polynomial-time performance, no more than a polynomial number of environments can be explored. Since environments are constructed beginning with the empty environment $\{\}$ and incrementally following the cost ordering, a good cost function for a problem is therefore one that induces a metric with the property that an acceptable solution set is at polynomial (with respect to the size of assumptions) distance away from the empty environment.

In many applications, we know *a priori* some characterization of the solution sets. For instance, we may know that at least one solution set comes from a small subset of assumptions; or there exists a solution environment with less than l assumptions. Consequently, good cost functions can be designed accordingly:

1. $\varrho(p) = 0$ if $p \subseteq \mathcal{A}'$, ∞ otherwise;

2. $\varrho(p) =$ *size-of*(p),

where the assumption sets is $\mathcal{A}$, the small subset of assumptions is $\mathcal{A}' \subseteq \mathcal{A}$, and $p \subseteq \mathcal{A}$ is any environment. Observe that the combination of these cost functions with the BF-ATMS algorithm subsume the focusing strategies described in Section 5.

8 DIAGNOSIS FROM FIRST PRINCIPLES

In [Reiter, 1987], Reiter described an algorithm for generating diagnoses in which conflicts of a system are generated incrementally. At each stage, one computes the hitting sets of the conflicts to find possible diagnoses of the system. However, missing in the paper is an efficient conflict generator. In addition to being an efficient (minimal) conflict generator, the BF-ATMS algorithm can also be used as an efficient consistency checker (see Section 3). In the following, we describe a diagnostic algorithm in which the BF-ATMS algorithm plays an important role.

As a motivating example, consider the circuit in Figure 1. Notice that the number of conflicts is 2^n even though there are only $2n + 1$ kernel diagnoses. Using a cost function similar to the above, one can efficiently find a conflict. On the other hand, to find the entire set of conflicts would have taken too much time. This clearly demonstrates that it is not a good idea for any diagnostic system, *e.g.* GDE [de Kleer and Williams, 1987b] and Sherlock [de Kleer and Williams, 1989], to take the approach of first generating the entire or a large collection of conflicts before the generation of diagnoses.

The following are the formal definitions relevant to our study of diagnosis. Most of them are borrowed directly from [de Kleer *et al.*, 1990].

Definition: A system is a triple (SD, COMPS, OBS) where:

- SD, the system description, is a set of first order sentences;
- COMPS, the system components, is a finite set of constants;
- OBS, a set of observations, is a finite set of first order sentences.

Definition: An AB-literal is AB(C), or $\neg$AB(C) for some $C \in$ COMPS. An AB-clause is a disjunction of AB-literals containing no complementary pair of AB-literals. A *conflict* of (SD, COMPS, OBS) is an AB-clause entailed by SD $\cup$ OBS. A *minimal* conflict is a conflict where no proper subset of it is a conflict.

We will adopt a result (Corollary 1) from [de Kleer *et al.*, 1990] as our definition of kernel diagnosis:

Definition: The *kernel* diagnoses of (SD, COMPS, OBS) are the prime implicants of the minimal conflicts of SD $\cup$ OBS.

Note that the prime implicants of the minimal conflicts are precisely the minimal hitting set of the minimal conflicts, *i.e.* the smallest sets in the cross product of the minimal conflicts. One usually multiplies the minimal conflicts in an *incremental* fashion in deriving the desired kernel diagnoses. The following definition allows us to describe such a process more precisely:

Definition: An *incomplete diagnosis* is a subset of a diagnosis. The *expansion* of an incomplete diagnosis ρ *by* a conflict C is the set of new incomplete diagnoses $\{\rho \cup \{c\} \mid c \in C\}$, while the process of deriving this set is called *expanding* the incomplete diagnosis ρ by C.

The process of generating diagnoses as described in [Reiter, 1987] is to: (1) start from the empty set as the only *incomplete* diagnosis, and (2) replace elements in the collection at each step by their *expansions* (with some suitable conflicts), until we obtain the desired diagnoses.

Using essentially the same approach, we consider the following algorithm for generating kernel diagnoses:

Procedure GEN-DIAG

1. Check if the current ATMS is consistent. If so, report that the only kernel diagnosis is the empty set and quit.
2. Initialize the incomplete diagnoses database to contain the empty list as the only element.
3. Retrieve an element ρ from the incomplete diagnoses database. If none exists, then quit.
4. Generate a minimal conflict C.
5. Consider every element in the expansion of ρ by C. If it is consistent with the system, insert it into the diagnoses database, otherwise insert it to the incomplete diagnoses database.
6. Repeat from step 3.

In this discussion, we will only be interested in systems that can be encoded in finite propositional Horn clauses for which we have an efficient conflict generator and consistency checker. In particular, in an actual implementation of the above algorithm, the BF-ATMS algorithm is used to check both the consistency for verifying a diagnosis and to generate a conflict for expanding an incomplete diagnosis. Note that the conflict returned by the BF-ATMS algorithm is always minimal.

A major difference between the GEN-DIAG algorithm and Reiter's *HS*-tree algorithm is that for each expansion of an incomplete diagnosis, we generate a new minimal conflict. This contrasts with using the same conflict to expand every incomplete diagnosis in Reiter's approach. At first glance, this appears to be a more expensive approach. But there are several reasons why we do this. In particular, we have a conflict generator which generates minimal conflicts with great efficiency. Furthermore, as we shall see later, we can focus the ATMS to return a good conflict for each incomplete diagnosis so that the search space for the diagnoses can be greatly pruned. Another important reason is that this incremental fashion of expanding incomplete diagnosis allows us to incorporate a cost function into the system so that lower-cost kernel diagnoses can always be generated before the consideration of the higher-cost ones, as in the case of label generation in the BF-ATMS algorithm.

From the definition of kernel diagnosis, we know that the GEN-DIAG algorithm will eventually find all the kernel diagnoses of the system. However, many improvements to it are needed to make it efficient. First, we need only to maintain a minimal set of incomplete diagnoses, *i.e.* all incomplete diagnoses that are subsumed by another incomplete diagnoses can be deleted from the incomplete diagnoses database. This is because every kernel diagnosis that will be generated (via expansion) by an incomplete diagnosis will also be generated by a smaller incomplete diagnosis. Similarly, since we are only interested in kernel diagnoses, we need only to maintain a minimal set of diagnoses in the diagnoses database. Therefore, the insert operation in step 5 of the above procedure should delete elements in the database which are subsumed by the new element or discard the new element if it is subsumed by an existing element.

A second improvement for the procedure can be derived from the observation that many conflicts generated in step 4 may not contribute to the process. In particular,

Proposition 10 *Suppose ρ is an incomplete diagnosis, if $\rho \subseteq \delta \subseteq$ COMPS, then δ is a diagnosis of* (SD, COMPS, OBS) *if, and only if, $\delta - \rho$ contains a hitting set of the set of conflicts S_ρ that has empty intersection with ρ. In addition, δ is kernel only if $\delta - \rho$ is a minimal hitting set of S_ρ.*

Proof: ($\Rightarrow$) Given that δ is a diagnosis, we know that δ is a hitting set of all conflicts. In particular, δ must hit every conflict in S_ρ. Since each conflict in S_ρ has empty intersection with ρ, $\delta - \rho$ must hit every conflict in S_ρ, *i.e.* $\delta - \rho$ contains a hitting set of S_ρ.

($\Leftarrow$) Given that $\delta - \rho$ contains a hitting set of S_ρ, by definition of S_ρ, we infer that δ must also be a hitting set of all conflicts, *i.e.* a diagnosis of (SD, COMPS, OBS).

Suppose δ is a kernel diagnosis of (SD, COMPS, OBS), if $\delta - \rho$ is not a minimal hitting set of S_ρ, then there must exists a $H \subset \delta - \rho$ that is a hitting set of S_ρ. But by definition of S_ρ, ρ itself is a hitting set of all other conflicts of (SD, COMPS, OBS), *i.e.* $H \cup \rho$ is a hitting set of all conflicts and hence a diagnosis. But this diagnosis is strictly smaller than δ which contradicts the assumption that δ is kernel. Hence, $\delta - \rho$ must be a minimal hitting set of S_ρ. □

The above result tells us that the ATMS should avoid considering any environment which contains an assumption representing an element of the incomplete diagnosis under consideration. This can be easily achieved by setting the labels for such assumptions to the empty set, thereby improving the efficiency of generating the next relevant conflict.

Another similar improvement is to realize that the kernel diagnoses which are already generated can also help to focus the BF-ATMS algorithm so that extraneous assumptions can be excluded from the conflict generation process. In particular,

Proposition 11 *Suppose we are expanding an incomplete diagnosis ρ and Δ is the current set of kernel diagnoses, then one can ignore the following set of assumptions from the conflict without affecting the results of expanding ρ:*

$$G = \{c \mid \exists \delta \in \Delta \ s.t. \ \delta - \rho = \{c\}\}$$

Proof: By the definition of G, any expansion of ρ by G will be a diagnosis subsumed by at least one of the kernel diagnoses already generated. Hence, such elements will be detected as diagnoses and discarded by the GEN-DIAG procedure. Therefore, we will not miss out any new kernel diagnoses if we ignore the assumptions in G. $\square$

The above result allows us to modify the ATMS as follows:

Proposition 12 *Assume the same suppositions as in Proposition 11. If we replace every assumption G in the ATMS by a premise, i.e. replacing the components set by* $\text{COMPS}' = \text{COMPS} - \{X \mid \text{AB}(X) \in G \text{ or } \neg\text{AB}(X) \in G\}$ *and the observations by* $\text{OBS}' = \text{OBS} \cup \{c \mid c \in G\}$*, then for every C which has empty intersection with ρ, we have C is a conflict of* $(\text{SD}, \text{COMPS}, \text{OBS})$ *if, and only if, $C \supseteq G$ and $C - G$ is a conflict of* $(\text{SD}, \text{COMPS}', \text{OBS}')$. *Furthermore, C is minimal if, and only if, $C - G$ is a minimal.*

Proof: For the first part,

($\Rightarrow$) Suppose C is a conflict of $(\text{SD}, \text{COMPS}, \text{OBS})$, first we want to show that $C \supseteq G$. For every $g \in G$, we know by definition of G that $\rho \cup \{g\}$ is a diagnosis. So the intersection of $\rho \cup \{g\}$ with C must be non-empty. But $\rho \cap C = \phi$, so g must be in C, *i.e.* $C \supseteq G$. We also know that

$$\text{SD} \cup \text{OBS} \cup \{c \mid c \in C\}$$

is inconsistent. Since $\{c \mid c \in C\}$ is already in OBS', we infer that

$$\text{SD} \cup \text{OBS}' \cup \{c \mid c \in C - G\}$$

is inconsistent. Furthermore, $\{X \mid \text{AB}(X) \in C - G \text{ or } \neg\text{AB}(X) \in C - G\} \subseteq \text{COMPS}'$, hence, $C - G$ is a conflict of $(\text{SD}, \text{COMPS}', \text{OBS}')$.

($\Leftarrow$) Suppose $C - G$ is a conflict of $(\text{SD}, \text{COMPS}', \text{OBS}')$, we know that

$$\text{SD} \cup \text{OBS}' \cup \{c \mid c \in C - G\}$$

is inconsistent. By swapping the set $\{c \mid c \in C\}$, we have

$$\text{SD} \cup \text{OBS} \cup \{c \mid c \in C\}$$

is inconsistent. Hence, C is a conflict of $(\text{SD}, \text{COMPS}, \text{OBS})$.

To prove the second part of the proposition, suppose $C - G$ is not minimal. Then there must exist $A' \subset C - G$ such that A' is also a conflict of $(\text{SD}, \text{COMPS}', \text{OBS}')$. Let $A = A' \cup G$, we know from the first part of the proposition that A must be a conflict of $(\text{SD}, \text{COMPS}, \text{OBS})$. But $C \supseteq G$ implies that $A \subset C$, so C is not a minimal conflict of $(\text{SD}, \text{COMPS}, \text{OBS})$.

Conversely, suppose C is not minimal, then there must exists $A \subset C$ such that A is a conflict of $(\text{SD}, \text{COMPS}, \text{OBS})$. From the first part of this proposition, we know that $A' = A - G$ must also be a conflict of $(\text{SD}, \text{COMPS}', \text{OBS}')$. But $C \supseteq G$ implies that A' must be a strict subset of $C - G$, *i.e.* $C - G$ is not a minimal conflict of $(\text{SD}, \text{COMPS}', \text{OBS}')$. $\square$

Corollary 13 *Suppose $\rho \subseteq \delta \subseteq$* COMPS *and $\delta \cap G = \emptyset$, then δ is a diagnosis of* (SD, COMPS, OBS) *if, and only if, δ is a diagnosis of* (SD, COMPS′, OBS′) *as defined in Proposition 12.*

Proof: We know from Proposition 10 that δ is a diagnosis of (SD, COMPS, OBS) if, and only if, $\delta - \rho$ contains a hitting set of S_ρ. Given that $\delta \cap G = \emptyset$, it means that δ does not hit any element in G. Therefore, δ is a diagnosis of (SD, COMPS, OBS) if, and only if, $\delta - \rho$ contains a hitting set of $\{C - G \mid C \in S_\rho\}$ which by Proposition 12 is exactly the conflicts of (SD, COMPS′, OBS′). Hence, δ is diagnosis of (SD, COMPS, OBS) if, and only if, δ is a diagnosis of (SD, COMPS′, OBS′). □

Since Proposition 11 ensures that we will not miss anything by ignoring the expansions which involve elements from G, we can restrict our consideration to diagnoses which have empty intersection with G. But by Corollary 13, we know that this is equivalent to considering the diagnoses of a reduced system in which components in G are deleted. Hence, at each stage of expanding an incomplete diagnosis ρ, we can replace all the assumptions representing the elements in G by premises, *i.e.* replace their labels by the singleton containing the empty set environment. This results in an ATMS with a smaller set of assumptions, and which generates smaller conflicts.

The above results guarantee that the new ATMS with lesser assumptions will still provide us with all the conflicts that are needed for the purpose of diagnosis computation. Thus, the BF-ATMS algorithm will be more efficient in conflict generation because of a smaller number of assumptions. Similarly, the generation of diagnoses from the smaller conflicts will also be more focused and efficient.

In an actual application, we may not want to generate every kernel diagnosis since this can be very expensive. Therefore, we can learn from our discussion on the BF-ATMS algorithm and impose a cost function to focus the diagnosis generation process. In particular, we can select a cost function which satisfies the monotonicity requirement and use it to control the order in which diagnoses will be generated. This can be achieved by modifying the incomplete diagnosis retrieval operation to always return a least-cost element. With this mechanism of generating diagnoses with least cost first, we can impose a cost bound to stop the system from considering elements in the expansion of an incomplete diagnosis with cost higher than the bound. In this case, all and only those kernel diagnoses with cost lower than the bound will be generated. We can also impose a stopping criterion to halt the algorithm when a desirable number of diagnoses has been obtained, *e.g.* when the probability mass of these diagnoses or the number of diagnoses exceeds a certain bound.

From the discussion above, we can modify the GEN-DIAG algorithm as follows:

Procedure GEN-DIAG

1. Check if the current ATMS is consistent. If so, report that the only kernel diagnosis is the empty set and quit.
2. Initialize the incomplete diagnosis database to contain the empty list as the only element.

3. Retrieve a lowest cost element ρ from the incomplete diagnoses database. If none exists, then quit.
4. Generate a minimal conflict C which shares no common component with ρ.
5. Consider every expansion of ρ by C that is within the cost bound; if it is consistent with the system, insert it into the diagnoses database, otherwise insert it into the incomplete diagnoses database.
6. If sufficient kernel diagnoses have been generated, then quit.
7. Repeat from step 3.

The above algorithm controls the diagnosis generation process by imposing a bound on the cost of a diagnosis and a criterion which indicates whether a sufficient number of diagnoses have been generated. Even though only the incomplete diagnosis with lowest cost is used for expansion each time, the diagnoses produced are not guaranteed to be the lowest cost among the remaining diagnoses. If it is a requirement for the kernel diagnoses to be generated monotonically with respect to their cost, we can either modify the GEN-DIAG algorithm to mimic the BF-ATMS algorithm by increasing the cost bound incrementally, or to delay the consistency check of the elements in the expansion (step 5) until they are retrieved from the incomplete diagnoses database (step 3). For the case where the cost of an incomplete diagnosis is its length, however, no modification to the GEN-DIAG algorithm is needed to guarantee that the lowest-cost diagnoses are always generated first.

An important efficiency issue which we have not discussed is the structure of the databases. In many problems, the incomplete diagnoses database can grow very large in size. It is therefore important to have an efficient data structure to support the insert operation which mainly involves subset checkings. A suitable data structure is the discrimination net described in [Forbus and de Kleer, 1992], which has very nice complexity behavior in the worst case situation.

As a conclusion to this section, we note that the major differences between our diagnostic algorithm and that of Reiter's are:

1. for each expansion of an incomplete diagnosis, we generate a new conflict,
2. the conflict generated is always minimal and shares no common literal with the incomplete diagnosis, and
3. the notion of cost is used to focus the system so that diagnoses with lower costs (or higher probabilities) are always generated first.

9 AN EXAMPLE

As an example, we use the circuit in Figure 1 to illustrate the efficiency of our cost-based approach. The BF-ATMS algorithm is controlled by a simple cost function, similar to the one described in Section 7, which prefers the assumptions that are different from $A_i, 1 \leq i \leq n$.

The cost function which controls the diagnosis generation process is simply the length of an incomplete diagnosis. An execution trace of the GEN-DIAG is shown in Figure 2, where (o), ($\surd$) and ($\times$) represent respectively the incomplete diagnoses, diagnoses and incomplete diagnoses that are subsumed by others, while [AB($\cdots$] represents the conflict generated for the incomplete diagnosis.

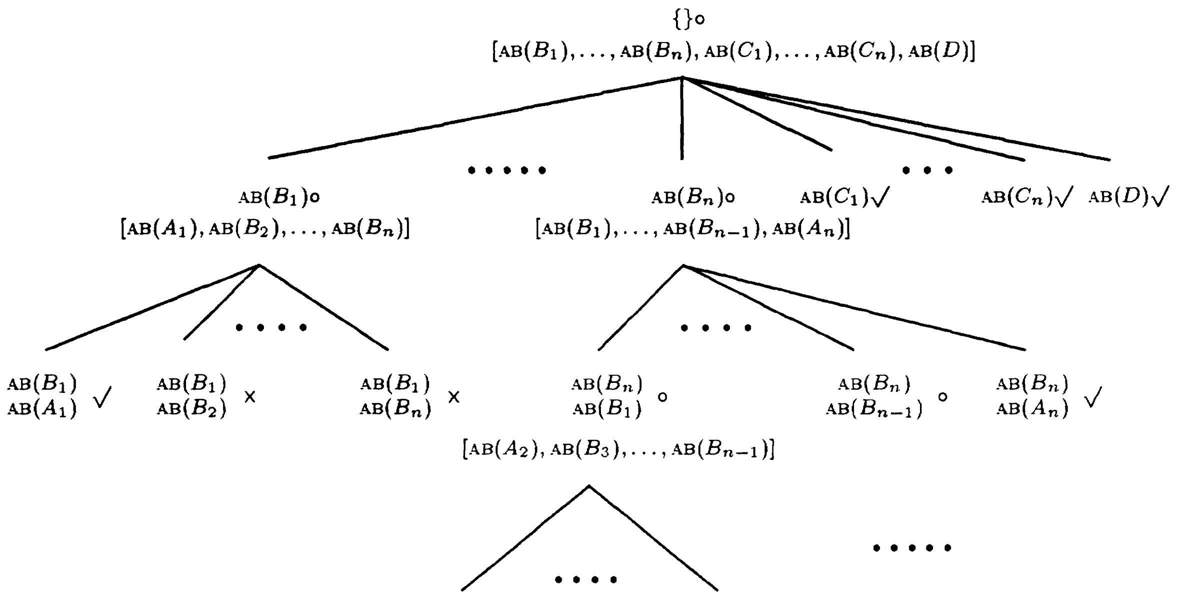
{}o
[AB(B_1), ..., AB(B_n), AB(C_1), ..., AB(C_n), AB(D)]
AB(B_1)o
[AB(A_1), AB(B_2), ..., AB(B_n)]
AB(B_n)o
[AB(B_1), ..., AB(B_{n-1}), AB(A_n)]
AB(C_1)$\surd$
AB(C_n)$\surd$
AB(D)$\surd$
AB(B_1) AB(A_1) $\surd$
AB(B_1) AB(B_2) ×
AB(B_1) AB(B_n) ×
AB(B_n) AB(B_1) o
[AB(A_2), AB(B_3), ..., AB(B_{n-1})]
AB(B_n) AB(B_{n-1}) o
AB(B_n) AB(A_n) $\surd$

Figure 2: The execution trace of the GEN-DIAG algorithm

First, the empty set was verified to be inconsistent with SD $\cup$ OBS and was initialized to be the only incomplete diagnosis. It was expanded by the conflict [AB(B_1), ..., AB(B_n), AB(C_1), ..., AB(C_n), AB(D)]. The incomplete diagnoses AB(C_1), ..., AB(C_n) and AB(D) were immediately checked out to be the single-fault diagnoses. Then the incomplete diagnosis AB(B_1) was expanded by the conflict [AB(A_1), AB(B_2), ..., AB(B_n)]. Note that the single-fault diagnoses were used to reduce the size of the conflict. Except for 'AB(B_1), AB(A_1)' which was checked out to be a diagnosis, every other element from the expansion was subsumed by an existing incomplete diagnosis and discarded. Other incomplete diagnoses are subsequently expanded in a similar fashion which produce additional diagnoses and incomplete diagnoses.

One important observation from this example is that the space of incomplete diagnoses which the algorithm explores is much smaller than the entire set of possible incomplete diagnoses. In particular, it only explores those of the form AB(B_{i_1}), ..., AB(B_{i_k}) where $1 \leq i_1 < \cdots, < i_k \leq n$ instead of the much larger collection AB(X_{i_1}), ..., AB(X_{i_k}) where $X \equiv A$ or $X \equiv B$.

In an experiment to see how a generate-and-test diagnostic system will perform, we modified the GEN-DIAG algorithm by expanding each incomplete diagnosis with the set of all assumptions except those useless ones as indicated in Propositions 10 and 11. An empirical comparison was made between the GEN-DIAG algorithm and the generate-and-test algorithm

GEN-TEST for the above example. The result is shown in Table 3, where *ID#* is the number of incomplete diagnoses that were expanded and *Time* is the execution time in seconds. It clearly demonstrates that the focusing power of the GEN-DIAG algorithm is superior in searching for multiple-fault diagnoses.

	Cost bound = 2				Cost bound = 4			
	GEN-DIAG		GEN-TEST		GEN-DIAG		GEN-TEST	
n	ID#	Time	ID#	Time	ID#	Time	ID#	Time
5	6	0.59	11	0.62	26	3.5	131	4.0
10	11	3.6	21	5.4	176	29.5	1161	226
15	16	13.6	31	19.7	576	286	4091	2260

Table 3: Empirical comparison between GEN-DIAG and GEN-TEST

For single-fault diagnoses, there is no significant difference between the two algorithms and either one may perform better than the other. For instance, if the conflict generated by the GEN-DIAG algorithm is not significantly smaller than the entire set of assumptions, then the cost of the conflict generation may outweigh its the focusing advantage. Conversely, if a small conflict is generated, then the additional cost of consistency checking incurred by the GEN-TEST algorithm may be more significant.

We have presented a diagnostic generator similar in spirit to Reiter's [Reiter, 1987]. Our new contributions include an efficient conflict generator and consistency checker, an efficient focusing technique for guiding the conflict generator in generating useful conflict and an incremental cost-based diagnoses generation process which prevents the system from wasting time on less probable diagnoses.

In a recent work [de Kleer, 1991], de Kleer has also implemented an incremental system that generates diagnoses in a best first fashion. This system differs from ours in terms of its focusing techniques. In particular, its conflict generator (HTMS) does not guarantee the conflicts it produces are minimal and thus can lead to additional cost for considering some useless incomplete diagnoses. Furthermore, the use of existing diagnoses in focusing the conflict generator is also lacking. On the other hand, de Kleer's work includes a minimum entropy technique for deciding which new probe can narrow down the space of probable diagnoses as quickly as possible.

10 RELATED WORK

We have described a general focusing method for the ATMS to solve problems efficiently. With a similar goal, some researchers have described other techniques for focusing the ATMS label updating process to avoid generating unwanted environments [de Kleer and Williams, 1987a; Forbus and de Kleer, 1988; Dressler and Farquar, 1990; Collins and DeCoste, 1991].

The incorporation of backtracking into an ATMS [de Kleer and Williams, 1987a] can be considered as a particular instance of a BF-ATMS where a special cost function is imposed on the literals in the task-specific control disjunctions. The backtracking procedure can roughly be simulated by a cost function which stipulates a preference ordering for the literals in each of the control disjunctions.

The approach taken by [Forbus and de Kleer, 1988; Dressler and Farquar, 1990] is to "focus" the ATMS to consider only small assumption sets at any one time and fix an upper bound on the length of environments. As described in Section 5, these are very effective techniques for controlling the label explosion problem. These focusing strategies, however, have no mean of "refocusing" the ATMS when the current focus is found to be unsatisfactory. Furthermore, the absence of the notion of a cost function encourages the creation of an *ad hoc* and unwieldy control structure in the problem solver itself. This removes one of the important advantages of using an ATMS: to alleviate the control burden from the problem solver.

In contrast, with an acceptable cost function for the problem domain, the BF-ATMS algorithm will independently generate the desirable low cost solution with greatly improved efficiency over the standard ATMS, without any intervention from the problem solver. Of course, one can also modify the BF-ATMS algorithm so that it can be interrupted, *e.g.* when the cost reaches a certain ceiling, and let the problem solver take over. The problem solver may in turn decide to run the algorithm with a better cost function. This is a much coarser-grained and more acceptable control than the previous approaches.

Another interesting approach to avoiding label explosion in an ATMS is to stop the label propagation at justified assumptions [Collins and DeCoste, 1991]. This approach appears to be very helpful for specific problems where many assumptions are each a consequence of some justifications. But as illustrated by some of the examples we gave earlier, there are problems where assumptions are never justified but still suffer from the label explosion problem. To make it worse, this explosion can still arise even if some of the assumptions are justified. In such cases, the avoidance of label propagation at justified assumptions does not help the combinatorial explosion.

This approach is related to, but different from, the simple application of heuristic search techniques [Pearl, 1984] to the ATMS. A fundamental problem addressed in this paper is how to merge the advantages of the efficiency of (a) heuristic search (exploration of a reduced portion of the search space to hit the solution faster) and (b) the ATMS (saving backtracking and redundant computation over a single-context TMS). It is the multiple-context search of the ATMS which makes the application of traditional heuristic search techniques to this task difficult. Most heuristic search techniques attempt to compute the single best solution, and keep a stack of next-best solutions if the current best partial solution becomes sub-optimal. Hence, for single-context searches, heuristic algorithms are easily defined. In contrast, the standard ATMS has no notion of best solution, only consistent or inconsistent solutions, and it computes all consistent solutions simultaneously.[7] Therefore, one needs to have a complete

[7]Here we loosely use solution for context. The ATMS maintains minimal representations of both consistent and inconsistent contexts, and the two are dual to one another. Strictly speaking, the ATMS can reason with either consistent or inconsistent minimal contexts. For example, the inconsistent contexts, besides performing the consistency maintenance function of an ATMS, are sometimes the desired solutions: in diagnosis, the

understanding of the fundamental steps of the ATMS algorithm (as in the BF-ATMS) before general focusing techniques can be devised without destroying the powerful properties of the ATMS.

11 CONCLUSIONS

This paper has described the application of a general notion of cost functions to the design of a new ATMS focusing algorithm. This successful incorporation of cost function into the ATMS algorithm is made possible because of the better understanding of an ATMS as described in Section 2.

The BF-ATMS algorithm focuses the label generation process to help the ATMS attain the efficiency of a traditional single-context TMS [Doyle, 1979; McAllester, 1985]. At the same time, it retains the multiple-context capability of an ATMS and the important properties of an ATMS like consistency, minimality, and soundness, in addition to the property of bounded completeness. Experimental results demonstrate that the BF-ATMS algorithm quickly solves problems proven to have labels which are of size exponential in the number of assumptions.

Moreover, the generality of the cost-function approach as well as the usefulness of the BF-ATMS algorithm are demonstrated by their applications to consistency-based diagnosis in which dramatic efficiency improvements, with respect to the simple generate-and-test technique, are obtained.

References

[Bodington *et al.*, 1990] R.M. Bodington, G.D. Sullivan, and K.D. Baker. Experiments on the Use of the ATMS to Label Features for Object Recognition. In *Proc. of ECCV-90*, pages 542–551, 1990.

[Brown *et al.*, 1987] Allen L. Brown, Dale E. Gaucas, and Dan Benanav. An algebraic foundation for truth maintenance. *Proc. of IJCAI-87*, pages 973–980, 1987.

[Chandra and Markowsky, 1978] Ashok K. Chandra and George Markowsky. On the number of prime implicants. *Discrete Mathematics*, **24**:7–11, 1978.

[Chang and Lee, 1973] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem-Proving.* Academic Press, 1973.

[Collins and DeCoste, 1991] John W. Collins and Dennis DeCoste. CATMS: An ATMS Which Avoids Label Explosions. In *Proc. of AAAI-91*, pages 281–287, 1991.

[de Kleer and Williams, 1987a] Johan de Kleer and Brian C. Williams. Back to backtracking: Controlling the ATMS. *Proc. of AAAI-87*, pages 910–917, 1987.

[de Kleer and Williams, 1987b] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, **32**:97–130, 1987.

inconsistent contexts are exactly the minimal conflicts used to generate the kernel diagnoses.

[de Kleer and Williams, 1989] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. *Proc. of IJCAI-89*, pages 1324–1330, 1989.

[de Kleer *et al.*, 1990] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses. *Proc. of AAAI-90*, pages 324–330, 1990.

[de Kleer, 1986a] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, **28**:127–162, 1986.

[de Kleer, 1986b] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, **28**:163–196, 1986.

[de Kleer, 1986c] Johan de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, **28**:197–224, 1986.

[de Kleer, 1991] Johan de Kleer. Focusing on probable diagnoses. *Proc. of AAAI-91*, pages 842–848, 1991.

[Dowling and Gallier, 1984] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, **3**:267–284, 1984.

[Doyle, 1979] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, **12**:231–272, 1979.

[Dressler and Farquar, 1990] O. Dressler and A. Farquar. Putting the Problem Solver Back in the Driver's Seat: Contextual Control of the ATMS. In *Second AAAI Workshop on Model-Based Reasoning*, pages 106–112, 1990.

[Dressler, 1990] Oskar Dressler. Problem solving with the NM-ATMS. In *Proc. European Conf. on AI*, pages 253–258, 1990.

[Forbus and de Kleer, 1988] Kenneth D. Forbus and Johan de Kleer. Focusing the ATMS. *Proc. of AAAI-88*, pages 193–198, 1988.

[Forbus and de Kleer, 1992] Kenneth D. Forbus and Johan de Kleer. *Building Problem Solvers.* To be published, 1992.

[Forbus, 1990] Kenneth D. Forbus. The qualitative process engine. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 220–235. Morgan Kaufmann, 1990.

[Gunter *et al.*, 1990] Carl A. Gunter, Teow-Hin Ngair, Prakash Panangaden, and Devika Subramanian. The common order-theoretic structure of version spaces and ATMS's. Technical Report MS-CIS-90-86, University of Pennsylvania, 1990.

[Kean and Tsiknis, 1990] A. Kean and G. Tsiknis. An Incremental Method for Generating Prime Implicants/Implicates. *Journal of Symbolic Computation*, **9**:185–206, 1990.

[Laskey and Lehner, 1990] K. Blackmond Laskey and P.E. Lehner. Assumptions, Beliefs and Probabilities. *Artificial Intelligence*, **41**:65–77, 1990.

[Lupanov, 1965] 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.

[McAllester, 1985] D. McAllester. A Widely Used Truth Maintenance System. Unpublished, 1985.

[McCarthy, 1980] J. McCarthy. Circumscription: A Form of Nonmonotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[Ng and Mooney, 1991] Hwee Tou Ng and Raymond J. Mooney. An efficient first-order horn-clause abduction system based on the ATMS. *Proc. of AAAI-91*, pages 494–499, 1991.

[Ngair, 1992] Teow-Hin Ngair. *Convex Spaces as an Order-theoretic Basis for Problem Solving.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1992.

[Orponen, 1990] P. Orponen. Dempster's Rule of Combination is #P-Complete. *Artificial Intelligence*, **44**:245–254, 1990.

[Pearl, 1984] Judea Pearl. *Heuristics : intelligent search strategies for computer problem solving.* Addison-Wesley, 1984.

[Provan, 1989] G. Provan. An Analysis of ATMS-based Techniques for Computing Dempster-Shafer Belief Functions. In *Proc. of IJCAI-89*, pages 1115–1120, 1989.

[Provan, 1990a] G. Provan. A Logic-based Analysis of Dempster-Shafer Theory. *International Journal of Approximate Reasoning*, Special Issue on Belief Functions and Belief Maintenance in Artificial Intelligence, 4:451–498, 1990.

[Provan, 1990b] G. Provan. *Complexity Analysis of Truth Maintenance Systems, with Application to High Level Vision.* PhD thesis, Faculty of Mathematics, University of Oxford, 1990.

[Provan, 1990c] G. Provan. The Computational Complexity of Multiple-Context Truth Maintenance Systems. In *Proc. of ECAI-90*, pages 522–527, 1990.

[Reiter and de Kleer, 1987] R. Reiter and J. de Kleer. Foundations of Assumption-based Truth Maintenance Systems: Preliminary Report. In *Proc. of AAAI-87*, pages 183–188, 1987.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, **13**:81–132, 1980.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, **32**:57–95, 1987.

[Wilson, to appear 1992] N. Wilson. Computational Efficiency and Generalisation of the Dempster-Shafer Theory. *Artificial Intelligence*, to appear, 1992.