Air Force Institute of Technology
AFIT Scholar

Theses and Dissertations | Student Graduate Works

12-1994

Generalized Probabilistic Reasoning and Empirical Studies on Computational Efficiency and Scalability

Eric P. Baenen

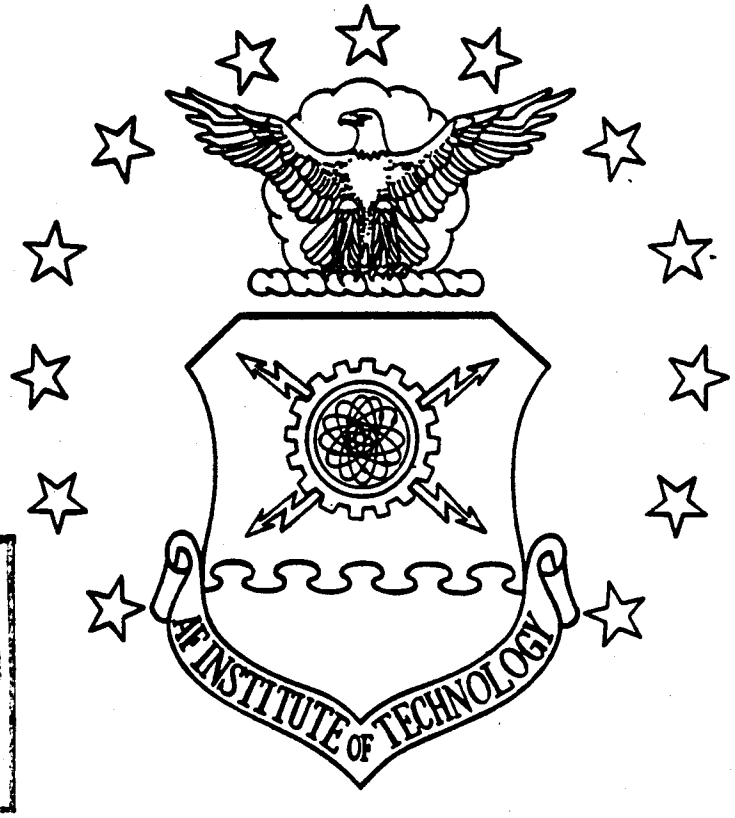
Follow this and additional works at: https://scholar.afit.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation

Baenen, Eric P., "Generalized Probabilistic Reasoning and Empirical Studies on Computational Efficiency and Scalability" (1994). *Theses and Dissertations*. 6369.
https://scholar.afit.edu/etd/6369

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.



DTIC
ELECTE
JUN 03 1994
F

Generalized Probabalistic Reasoning
and
Empirical Studies on Computational Efficiency and Scalability

THESIS
Eric Paul Baenen
Captain, USAF

AFIT/GCE/ENG/94D

19941228 008

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/94D

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Generalized Probabalistic Reasoning
and
Empirical Studies on Computational Efficiency and Scalability

THESIS

Eric Paul Baenen
Captain, USAF

AFIT/GCE/ENG/94D

DTIC QUALITY INSPECTED 2

Approved for public release; distribution unlimited

AFIT/GCE/ENG/94D

Generalized Probabalistic Reasoning

and

Empirical Studies on Computational Efficiency and Scalability

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Eric Paul Baenen, B.S.E.E.

Captain, USAF

December 1994

Approved for public release; distribution unlimited

Acknowledgements

I would like to dedicate this work to my parents, [redacted], my brother [redacted] and my sister [redacted], without whose support and encouragement it would not have been possible. I would also like to thank my close friends Melanie Steckbauer, Kim Zimmerman, and fellow students John Berry, Greg Alhquist, Terry Wilson, Tevis Boulware and many others too numerous to mention. Their support, encouragement and comradery each day made AFIT bearable.

Special thanks to Ed Williams and the other readers who provided valuable comments and suggestions that helped make this a better paper.

Last but not least I would like to thank my advisor and friend, Dr. Eugene Santos. I owe a great deal to his guidance and encouragement.

This research was supported by AFOSR Project 940006.

Eric Paul Baenen

Table of Contents

List of Figures

Abstract

Expert Systems are tools that can be very useful for diagnostic purposes, however current methods of storing and reasoning with knowledge have significant limitations. One set of limitations involves how to store and manipulate uncertain knowledge: much of the knowledge we are dealing with has some degree of uncertainty. These limitations include lack of complete information, not being able to model cyclic information and limitations on the size and complexity of the problems to be solved. If expert systems are ever going to be able to tackle significant real world problems then these deficiencies must be corrected. This paper describes a new method of reasoning with uncertain knowledge which improves the computational efficiency as well as scalability over current methods. The cornerstone of this method involves incorporating and exploiting information about the structure of the knowledge representation to reduce the problem size and complexity. Additionally, a new knowledge representation is discussed that will further increase the capability of expert systems to model a wider variety of real world problems. Finally, benchmarking studies of the new algorithm against the old have led to insights into the graph structure of very large knowledge bases.

Key Words: Expert System, Linear Constraint Satisfaction, Efficiency, Scalability, Inferencing, WAODAG, Large Knowledge Base Structure, Knowledge Representation, Bayesian Forest.

Generalized Probabalistic Reasoning

and

Empirical Studies on Computational Efficiency and Scalability

I. Introduction

1.1 Background

Knowledge Based Expert Systems (KBESs) are useful tools for aiding decision making, diagnosis, or process monitoring [44]. The expert system can be thought of as a framework consisting of three primary parts, the knowledge base, the user interface and an inference engine. The knowledge base is the collection or database of knowledge within a narrow problem domain such as medical diagnosis of blood disorders or infectious diseases [41, 3, 40], prospecting for oil [12], modeling chemical interactions [25] or configuring computers [26]. The inference engine is the portion of the KBES that manipulates this knowledge in order to solve our problem [16].

Almost every domain that is of interest to expert systems developers contains knowledge that is inherently inexact, incomplete or immeasurable. The method used to manipulate and combine this information is important in order to avoid results that are inaccurate and counter intuitive. A great deal of controversy exists concerning which method is the best for modeling and reasoning with uncertainty [11, 30, 46]. The community is very polarized, each method's proponents feel that their method is the only viable solution. [30, 43].

The reasoning model is the method we use to represent the knowledge so that it can be manipulated to perform some task. The choice of our representation will affect or determine which methods of inferencing we can use. The uncertainty of the knowledge deals with the truth value of a fact or set of facts which are not entirely true or entirely false, but instead lie somewhere in between. In other words there is some level or scale of truthfulness.

Uncertainty is most often the result of inaccurate or incomplete information but can also be caused by reasoning with and combining incomplete information. Another important source is the inaccurate and conflicting combination of knowledge from multiple experts [11, 17, 29, 30, 46].

Three of the most common methods of representing and reasoning with uncertain knowledge are Bayesian Networks [30], Dempster-Shafer Theory [11, 37], and Fuzzy Logic [46, 45] each with their own strengths and weaknesses [17, 29]. A newer representation called a Weighted AND OR Directed Acyclic Graph (WAODAG) [6] is much more versatile in generally representing real world knowledge and combined with a method of converting the graph into a system of linear inequalities [34, 22, 36] should be much faster in solving for the optimum solution of a problem [34].

1.2 Problem

The purpose of this research is to determine an inferencing methodology that will allow us to solve currently modeled problems more efficiently. We also want to be able to solve larger and more complex problems than were previously possible.

The inferencing methodology must consist of two parts, the representational model and the method of extracting inferences from that representation.

We will investigate new models of representing and reasoning with uncertain knowledge, such as WAODAGs, that resolve some of the problems and limitations of current reasoning methods. Specifically we want to address the problems of having incomplete information about the uncertainties we are reasoning with, of not being able to incorporate cyclic information, and limitations on the size and complexity of the problems we can attack. If we are ever going to be able to create systems that reason about significant real world problems and solve them in a reasonable amount of time, these are deficiencies that must be addressed.

In real world problems, we rarely if ever have complete knowledge about a situation. This results in gaps in our information. Information about the actual uncertainties is no different. For instance, in a medical diagnosis the doctor may have evidence that supports the

diagnosis of a particular disease. Lack of that evidence, however, might not lend support to the diagnosis that the patient does not have the disease.

Cyclic information is important because many of the processes that occur in nature are cyclic. For instance the simple representation of fire implies smoke and smoke implies fire cannot be represented in current methods. Current graph traversal techniques for inferencing require that the graphs be acyclic or they face the danger of infinite looping in the cycle.

Perhaps one of the most significant deficiencies in current approaches is the size and complexity of the problems they are able to manipulate. If a user has to wait too long for the results (too long being dependent upon the application the expert system is being used for) they are unlikely to continue using the tools. In addition, if we are only able to solve small problems then some domains which could greatly benefit from such tools will be consistently out of reach of this technology, particularly when the problem cannot be broken down sufficiently.

For example, the WAODAG representation is more versatile in representing uncertain and partial knowledge than other representations, however the algorithm that solves a WAODAG is currently limited in the size and complexity of the problems it is able to solve. The WAODAG representation, like the other models, also cannot deal with cyclic information.

1.3 Scope

This study was limited to probabilistic models of representing uncertain information. Other models exist, however they lack the structured methodology and implementation of the probabilistic models as well as the tools for analysis.

1.4 Approach

Earlier work with WAODAGs [34] demonstrated that inferencing could be accomplished by converting the WAODAG graphs into boolean linear programs [34, 36] which were then solved using techniques from the field of Operations Research such as the Simplex method [22, 27, 36, 28, 34]. The earlier methods while effective, are somewhat crude and limited in the size and complexity of problems they can solve. In addition, the Simplex method is an

algorithm for optimizing arbitrary sets of linear inequalities. It does not care where they came from or what they represent. It is our belief that we can take advantage of information about the structure of our representation and incorporate this knowledge into the equations to reduce the size and complexity of the problems that are fed to the Simplex method thereby reducing the amount of time necessary to solve for the optimum or most probable solution.

The approach taken was to develop a new algorithm for inferencing with WAODAGs that incorporates and exploits knowledge of its structure to reduce the problem size and complexity. This should substantially increase the computational efficiency over the old algorithm as well as make it possible to solve larger and more complex problems. A new knowledge representation called a Bayesian Forest [34, 31] will then be explored to see if that representation is able to resolve some of the deficiencies that still exist with the WAODAG representation and to see if the new reasoning algorithm can be generalized to work with the new knowledge representation.

1.5 Solution

First, we developed and extensively studied an efficient algorithm for inferencing with WAODAGs which incorporates knowledge of the structure of the graphs into all phases of the problem. We used WAODAGs because of their versatility and subsumption of other models such as Bayesian networks [6].

We found that our solution is much better than existing algorithms. Most significantly the algorithm is faster at finding the optimum solution as well as being able to solve much larger problems than could ever be tackled before. Benchmarking of the new algorithm against the old emphasized testing with realistic large random graph structures which led to an exploration of large knowledge base graph structures. This study yielded important insights into what the structure of very large real world knowledge bases might look like.

Additionally, we explored the new model for representing uncertainty called a Bayesian Forest [34, 31] that overcomes many of the limitations and deficiencies of the other current models. Models such as the Bayesian network can even be converted into Bayesian Forests

with no loss of information since Bayesian networks, like WAODAGS, are actually a subset of Bayesian Forests. These Bayesian Forests can use either the current or new method of inferencing and promise to efficiently solve larger and more complex problems than are possible with current models and methods.

1.6 Executive Overview

We have successfully shown that information about the structure of a knowledge representation can be incorporated into the problem in order to more efficiently solve for the optimum solution. This research resulted in a new algorithm for inferencing with either WAODAGs or Bayesian Forests that is more efficient and can solve larger and more complex problems than were possible before. The Bayesian Forest knowledge representation was examined and found to be a more complete and versatile representation that actually subsumes both the WAODAG and Bayesian network representations. It was found that the research using the WAODAG representation can be generalized to work with the Bayesian Forest representation. An efficiency study of the reasoning algorithms led to important insights into what the structure of very large real world knowledge bases might look like. There are indications that very large scale knowledge bases are likely subdivided into smaller loosely connected cells that are themselves loosely connected. There are also indications that there might be some general limit to the depth of a larger knowledge base, probably in the neighborhood of 5 to 8 levels of causality.

II. Literature Review

In this chapter we will discuss the three methods of manipulating uncertain knowledge used most commonly in expert systems: Bayesian Networks [30], Dempster Shafer Theory [11, 37] and Fuzzy Logic [46, 45]. We will describe the basis of each method as well as some of their inherent advantages and disadvantages. Finally we will discuss a relatively new method of representing [6] and manipulating [34] uncertain knowledge called a WAODAG [6]. This method provides an efficient means of reasoning while eliminating some of the weaknesses of the previous methods [34].

2.1 Bayesian Networks

Bayesian belief networks are rooted in traditional subjective probability theory which builds on the foundation of Pascalian calculus. In subjective probability theory the probability of a proposition represents the degree of confidence an individual has about that proposition's truth. This matches quite well to our knowledge base of information from a human expert in addition to his or her subjective beliefs about the accuracy of that information [29, 30].

Before we can describe Bayesian belief networks we must begin with the fundamentals of probability theory. Let A be some event within the context of all possible events E, within some domain, such that $A \in E$ and E is the event space. The probability of A occurring is denoted by $P(A)$. $P(A)$ is the probability assigned to A prior to the observation of any evidence and is also called the *apriori* probability. This probability must conform to certain laws. First, the probability must be non-negative and must also be less than one, therefore

$$\forall A \in E, 0 \leq P(A) \leq 1 \tag{1}$$

A probability of 0 means the event will not occur while a probability of 1 means the event will always occur. Second, the total probability of the event space is 1 or in other words

the sum of the probabilities of all of the events A_i in E must equal 1.

$$\forall A \in E, \sum A_i = 1 \tag{2}$$

Finally, we consider the compliment of A, $\neg A$, which is all events in E except for A. From equation 2 we then get

$$P(A) + P(\neg A) = 1 \tag{3}$$

Now consider another event in E, B such that $B \in E$. The probability that event A will occur given that event B has occurred is called the conditional probability of A given B and is represented by $P(A \mid B)$. The probability that both A and B will occur is called the joint probability and is defined by $P(A \cap B)$. $P(A \mid B)$ is defined in terms of the joint probability of A and B by

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{4}$$

Equation (4) can be further manipulated to yield Bayes Rule

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)} \tag{5}$$

If these two events are independent, in that the occurrence of one event has no effect on the occurrence of the other, then $P(A \mid B) = P(A)$ and $P(B \mid A) = P(B)$. If we manipulate equation 5 still further we get

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{[P(B \mid A) \times P(A)] + [P(B \mid A) \times P(\neg A)]} \tag{6}$$

This lays the foundation for managing and manipulating uncertainty using probability theory in expert systems. It allows us to turn a rule around and calculate the conditional probability of A given B from the conditional probability of B given A. For example consider the rule, if hypothesis A is true then some evidence B will be observed with a

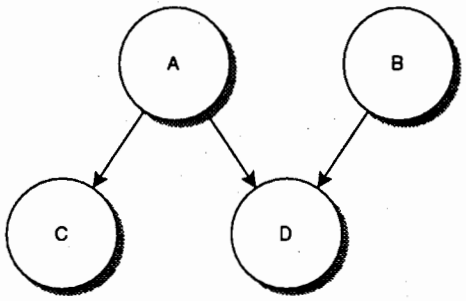


Figure 1. A small Bayesian network.

probability of p. Without knowing the truth value of A, if B is observed then from the previous equations we can calculate the probability that hypothesis A is true [4, 29, 30].

Bayesian belief networks use this information in a graphical form, an acyclic directed graph, to represent and manipulate uncertain knowledge. Nodes in the graph represent states or events and have some conditional probability of occurrence associated with them. The arcs connecting the nodes represent the conditional dependencies between nodes. Each arc may only connect two nodes and must be unidirectional. Figure 1 presents a small example network with only four nodes. In this example the state of node C is dependent on the state of node A, and the state of node D is dependent on the states of both nodes A and B. Nodes A and B are independent nodes if we are given D. Nodes C and D are independent of one another if we are given A and B. Therefore to determine the conditional probability of D we need only concern ourselves with the states of A and B since $P(D \mid A, B, C) = P(D \mid A, B)$. We can also determine from the graph that $P(C \mid A, B, D) = P(C \mid A)$. In order to determine the probability of a possible world where each node has a set value, for example A and C are True and B and D are False, we must have all of the conditional probabilities from the following list.

$$P(A) = p_a \qquad P(B) = p_b$$
$$P(C \mid A) = w \qquad P(C \mid \neg A) = x$$

$$P(\neg C \mid A) = y \qquad P(\neg C \mid \neg A) = z$$
$$P(D \mid A, B) = a \qquad P(D \mid A, \neg B) = b$$
$$P(D \mid \neg A, B) = c \qquad P(D \mid \neg A, \neg B) = d$$
$$P(\neg D \mid A, B) = e \qquad P(\neg D \mid A, \neg B) = f$$
$$P(\neg D \mid \neg A, B) = g \qquad P(\neg D \mid \neg A, \neg B) = h$$

The number of probabilities for each dependent node is equal to 2^n where n is the number of conditional dependencies for the node [29, 30, 35]. To find the probability of a particular world we must have a complete assignment to all nodes in the graph. The conditional probabilities from the table for the appropriate instantiations are then multiplied together to give the probability of the world. In the previous example, the probability of the world would be $w \times f$ if we are given A is true and B is false. The object of inferencing is to find the world with the maximum probability given the evidence.

Some of the advantages of Bayesian belief networks are that the representation is visual and easy to understand. It is also relatively straight forward to implement as the methodology for combining uncertainty follows set rules and procedures. Probability theory is a well-refined method for dealing with knowledge of unknown certainty [29, 30].

Bayesian belief networks still have some problems. They require large numbers of probabilities that must be obtained from the human expert. The number of probabilities is dependent on the complexity of the conditional dependencies in the domain. They also cannot represent cycles (eg. A implies B and B implies A) or infinite loops would occur during inferencing. Additionally because the sum of all possible states must equal 1, when evidence reinforces the belief in some possible world, it correspondingly decreases our belief in all other worlds. This is not necessarily the case in real life. Consider the case of medical diagnosis. A positive result on some test may increase our belief that the patient has some malady, however it does not necessarily decrease our belief that the patient has any other disease. It may in fact have no effect at all on our belief in another possible world. Finally, Bayesian networks

require us to make certain artificial assumptions about the independence of information/events leading to counter intuitive, possibly incorrect results [18, 24, 21, 29, 30, 13, 35].

2.2 *Dempster-Shafer Theory*

Dempster-Shafer Theory (DST) was started by Arthur Dempster in the 1960's and expanded by Glen Shafer in the 1970's [11, 37]. Dempster felt there was a need for a new system of dealing with uncertainty because of two shortcomings he saw with probability theory. The first problem is the difficulty of representing ignorance. In probability theory ignorance is represented by uniform probabilities. To some, this approach seems to imply more information than was given, since equal prior beliefs can be attributed to either complete ignorance or to an equal belief in all hypothesis. The other problem Dempster recognized with probability theory was the idea that the subjective belief in an event and its negation must sum to one. He claimed that in many situations evidence that supports one hypothesis should not necessarily decrease the belief in all others [11, 29, 37].

Dempster-Shafer theory represents ignorance explicitly by working with the power set of all possible hypothesis within the domain. It also does not fix the probability of the negation of a hypothesis once the probability of the hypothesis itself is known.

Like probability theory DST starts with a universe of possible hypotheses, called the frame of discernment. However, in DST that universe contains the power set of those possible hypotheses and that is its foundation. In a universe U with possible singleton hypotheses A, B, and C such that A,B, and $C \in U$, there are 9 possible hypotheses including $\{A\}$, $\{B\}$, $\{A, B\}$, and $\{A, B, C\}$. Ignorance is represented by hypothesis sets that contain more than one element. If the focal elements, subsets of U with nonzero basic probability assignments, are all singletons, then no ignorance exists regarding their occurrences. Further more, in DST evidence against a hypothesis only supports the negation of that hypothesis. For example, evidence against the hypothesis $\{A\}$ (A and only A) supports its negation $\{B, C\}$ (everything but A) but does not effect the other hypotheses sets such as $\{A, B\}$, $\{B\}$, or $\{A, C\}$ [9, 11, 29].

If we now consider the subset A of U, the probability assigned to the set A is represented by $m(A)$, which is the portion of total belief that has been assigned to A. An additional notation is $Bel(A)$ which is a measurement of the total belief in A. For singletons $Bel(A) = m(A)$, however for sets that contain more than one element this measurement is greater than or equal to m. For example, $Bel(\{A, B\}) = m(\{A, B\}) + m(\{A\}) + m(\{B\}) \geq m(\{A, B\})$ [9, 11, 29].

When the available evidence yields more than one belief measurement about a single hypothesis, the beliefs should be combined to form a single overall belief in the hypothesis. DST does this by computing their orthogonal sums with Dempster's rule of combination [11, 29].

Although Dempster-Shafer theory overcomes the representational difficulties of probability theory concerning ignorance and disconfirming the negation of a hypothesis it suffers from implementational complexity. The frame of discernment requires the exhaustive enumerating of all possible set of hypotheses within the domain universe. More importantly, DST lacks an effective methodology for extracting inferences from the belief functions. The result is that very few expert systems have ever been built using DST [9, 29, 38].

2.3 Fuzzy Logic or Possibility Theory

Fuzzy Theory or Possibility Theory was first developed by Lotfi Zadeh in the mid 1960's as an alternative to probability theory for representing and dealing with vague or imprecise information [46, 45]. In a knowledge base much of the information from the human experts is laced with phrases such as "very likely," "probably," and "sort of." When these terms are translated into subjective numeric probabilities the "fuzziness" is lost. Zadeh developed fuzzy logic to accurately manipulate this vague knowledge without loss of information. To do this the traditional binary logic of probability theory is replaced with a multivalued logic [46]. In a typical knowledge base of rules, with their antecedents and consequents, approximate matching of facts to a rule's antecedent is difficult or impossible with conventional two valued logic but becomes natural for multivalued fuzzy logic [29, 46].

Fuzzy logic starts with some set of objects, U. If A is a fuzzy subset of U then there is some function $\mu_A(u)$ which maps the elements of U into A by some number between 0 and 1. This number represents the degree of membership of the element in the set A. The difference from normal set theory is that rather than belonging or not belonging to a set, the elements of U can partially belong. If $\mu_A(u) = 1$ then membership is absolute and $\mu_A(u) = 0$ indicates nonmembership. When English modifiers are encountered in a rule base, these formula are altered using particular modifier rules to indicate increases or decreases in set membership [17, 29].

Inferencing in possibility theory is performed using generalized modus ponens. In standard modus ponens, if "$A \rightarrow B$" and "A" are true then "B" is true. In generalized modus ponens matching is not required to be exact and predicates can by fuzzy. For example, if A, A^*, B and B^* are fuzzy statements and we have the rule "If X is A Then Y is B" then given "X is A^*" implies Y is B^*." [17, 29]

Possibility theory enables inherently fuzzy knowledge to be represented and manipulated explicitly and easily with no loss of information. It does suffer from its share of problems however. Some properties of fuzzy sets have the potential to create inconsistencies when reasoning with fuzzy information. Another problem is the inherent lack of formal definitions for functional modifier rules. This can lead to inconsistencies between knowledge bases. Some subtle information can also be lost when similar linguistic terms are assigned the same modifier function. Perhaps the most important problem is that possibility theory lacks formal semantics [7, 17, 29]. These problems have not stopped many from creating successful expert systems [2, 14] particularly in Japan where fuzzy logic is widely used in a variety of fields [23, 29].

2.4 Weighted AND OR Acyclic Directed Graphs

Another possible knowledge representation is called a Weighted AND/OR Directed Acyclic Graph (WAODAG) [6]. Like the Bayesian belief network this is a graphical knowledge representation. An example of a small WAODAG is shown in Figure 2. The graph is

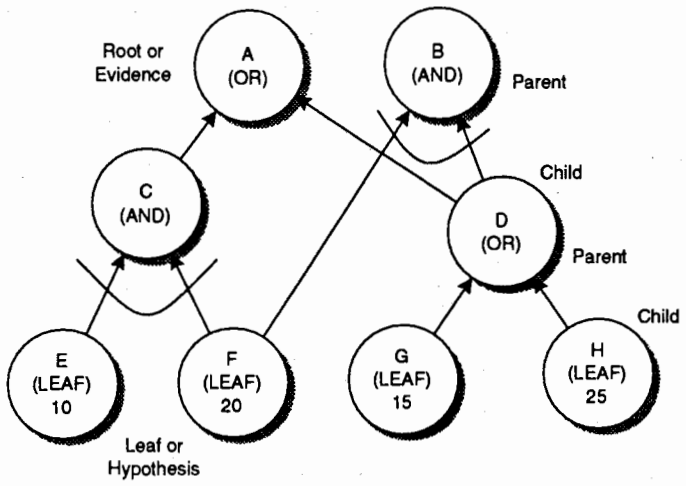


Figure 2. A simple Weighted AND OR Directed Acyclic Graph or WAODAG.

made up of nodes and interconnections or edges. The nodes represent facts or propositions in our knowledge base and can be either true or false. If a node is clamped or set to some value before the inferencing process begins then it is considered evidence and signifies what we are trying to explain. The edges represent relationships or dependencies between the propositions. The nodes come in three flavors, AND, OR, and LEAF nodes. An AND node is only true if all of the nodes it is dependent on, its child nodes, are true. An OR node is only true if one or more of the nodes it is dependent on is true. The LEAF nodes are those nodes that have no children and represent the possible hypotheses explaining the evidence. Originally each node has a cost which is the logarithm of the probability for that rule. It has been shown in [6] that the costs on the individual nodes can be combined and settled down to the LEAF nodes without loss of generality. The cost at the LEAF represents the cost incurred if this node is determined to be true. Summing the costs of the LEAF nodes that have been determined to be true, to find the cost of the particular world is equivalent to multiplying their probabilities [6]. The process of inferencing attempts to determine the best cost assignment of nodes in the graph to explain the evidence.

This representation works very well for modeling knowledge bases composed of if-then rules, one of the easiest and most flexible ways for knowledge engineers to codify the knowledge in a particular domain [42]. Figure 3 demonstrates how we convert two different types of rules into their corresponding representation for the WAODAG.

1: IF Event A occurs OR Event B occurs THEN Event C will occur with a Probability of P(C|A) + P(C|B)

2: IF Event A occurs AND Event B occurs THEN Event C will occur with a Probability of P(C|A,B)

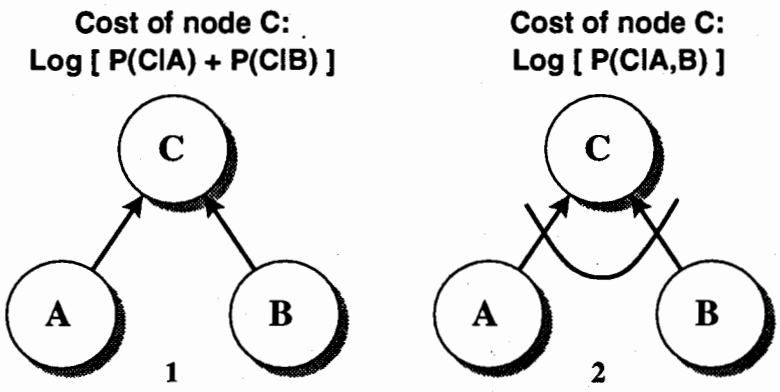


Figure 3. Converting rules from a rule base into nodes and edges for a WAODAG.

A flexible and structured knowledge representation is only half of the solution however. An efficient algorithm for inferencing over the representation must also be present for an effective and usable system. [34] has developed such a system. The algorithm converts the graphical structure of the WAODAG into a series of linear inequalities. This system of equations can then be solved using efficient linear programming techniques such as the Simplex method from the field of Operations Research [22, 28, 36, 27].

In order to convert the graph into the system of equations we let the true and false states of the nodes be represented by 1 and 0 respectively. We then let x_n denote a variable representing the state of node n. If n is true, $x_n = 1$ and if n is false $x_n = 0$. Further, we let

D_n represent the set of all nodes that are children of node n, S is the set of all evidence nodes and L_n is the set of all leaf nodes.

From [34] the complete set of equations representing the graph are:

$$\text{if q is an AND node:} \quad \forall p \in D_q \quad \{x_q \leq x_p\} \tag{7}$$

$$\text{if q is an AND node:} \quad \sum_{\forall p \in D_q} x_p - \mid D_q \mid +1 \leq x_q \tag{8}$$

$$\text{if q is an OR node:} \quad \sum_{\forall p \in D_q} x_p \geq x_q \tag{9}$$

$$\text{if q is an OR node:} \quad \forall p \in D_q \quad \{x_q \geq x_p\} \tag{10}$$

$$\text{Evidence:} \quad \forall q \in S \quad x_q = 1 \tag{11}$$

$$\text{Cost function:} \quad \sum_{q \in L_q} cost_{x_q} \times x_q \tag{12}$$

Equations 7 and 9 comprise the backward chaining constraints since they dictate propagation from the direction of the evidence while equations 8 and 10 represent the forward chaining constraints and dictate propagation from the direction of the causes or hypotheses [34]. Equation 12 is the cost function that the Simplex method will attempt to minimize in order to determine the most probable explanation for the evidence. From our example WAODAG in Figure 2, Figure 4 shows the system of linear inequalities created to describe that graph.

For the Simplex algorithm to solve them the first four equations are manipulated into the following form:

$$\{x_q - x_p \leq 0\} \, \forall p \in D_q \tag{13}$$

$$x_q - \sum_{\forall p \in D_q} x_p \geq 1 - \mid D_q \mid \tag{14}$$

$$x_q - \sum_{\forall p \in D_q} x_p \leq 0 \tag{15}$$

$$\{x_q - x_p \geq 0\} \, \forall p \in D_q \tag{16}$$

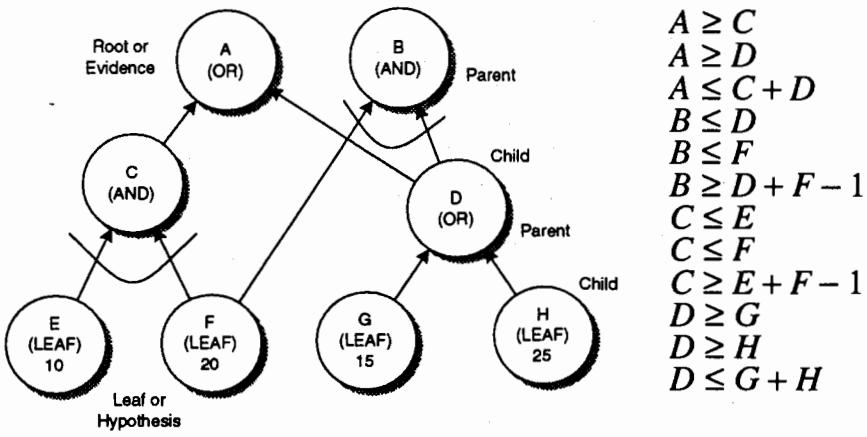


Figure 4. The simple WAODAG and the equations describing it.

Solving the equations using the Simplex algorithm involves a two stage process. The first phase of the Simplex algorithm finds a possible solution in the feasible space of solutions. The second phase of the Simplex algorithm finds the optimum solution. This two stage process is guaranteed to find the optimum solution [34, 22, 27, 36]. One problem arises in that the optimum solution using this method may not be an integral solution; or rather, does not correspond to a 0-1 assignment or mapping of values to the nodes. Because an assignment of 1 to a node indicates it is true and a 0 indicates it is false, a complete and valid assignment to the graph must be an integral 0-1 mapping. In the case of a non-integral solution the algorithm must go through an iterative branch and bound process to find the best integral solution. Research has shown that the entire algorithm exhibits an expected case polynomial time growth rate instead of the exponential growth rate of current search techniques [34].

2.5 *Summary of methods*

Each of the previous methods has its advantages and disadvantages. To quickly summarize:

2.5.1 Bayesian Belief Networks. The advantages of Bayesian Belief Networks are that it is a graphical representation that is easy to visualize and relatively straight forward to implement as well as having strong semantics and proven techniques for manipulation of information to calculate probabilities. The primary disadvantage is that it requires extensive probability tables. In fact inferencing has been shown to be an NP-Hard process [15, 8, 10, 39]. Other disadvantages include that it cannot represent cycles and requires certain possibly artificial independence assumptions. Finally there is the problem that all probabilities must sum to 1 which requires that evidence supporting belief in one possible world must decrease the belief in all other possible worlds whether correct or not.

2.5.2 Dempster-Shafer Theory. The advantages of Dempster-Shafer Theory lie in its ability to better represent ignorance as well as its structure allowing evidence supporting one possible world to not necessarily detract from belief in all other worlds. The disadvantages occur because of its implementational complexity and the requirement for exhaustive enumeration of all possible combinations of hypotheses. Dempster Shafer Theory also lacks an effective methodology for extracting inferences.

2.5.3 Fuzzy Logic or Possibility Theory. The advantages of Fuzzy Logic are that it not only models uncertainty well but represents and manipulates this fuzzy knowledge explicitly. In addition this representation makes it very easy for knowledge engineers to codify the knowledge in a particular domain. Unfortunately fuzzy logic lacks semantics for efficient inferencing. There is also the potential for inconsistencies due to inherent properties as well as a general lack of formal definitions for functional modifier rules.

2.5.4 Weighted AND OR Acyclic Directed Graphs. Like Bayesian networks, Weighted AND/OR Acyclic Directed Graphs are a graphical and intuitive knowledge representations that translates easily from a rule base. One of the most significant advantages of using the method of converting the graph to a system of linear inequalities and solving this linear program is that the algorithm exhibits an expected case polynomial time growth

rate [34]. In addition, the use of the probabilities associated with the rules provides strong semantics to the costs used in the representation [6]. While the disadvantages are that it still cannot deal with cyclic information and the current method of converting and solving the linear programs limits the size of the problem that can be solved.

III. Methodology

In this chapter we will discuss how we can improve computational efficiency in the original reasoning algorithm by incorporating information about the structure of our knowledge representation. By incorporating structural information into the problem before it is presented to the inferencing mechanism we hope to reduce the problem size and complexity, thereby reducing the amount of time necessary to solve for the best or most probable solution.

3.1 Reasoning Algorithm Enhancement

The original reasoning algorithm developed in [34] converts a directed acyclic graph into a system of linear inequalities. It then solves these equations using the Simplex method in a two stage process. The first phase of the Simplex algorithm determines an initial solution in the feasible space. The second phase of the Simplex algorithm finds the optimal solution. If the optimal solution is not an integral solution the algorithm must utilize a branch and bound process to find the optimal integral solution. This process is depicted in Figure 5.

The problem with the Simplex method is that it is an algorithm for optimizing arbitrary sets of linear inequalities. It blindly solves for the optimal solution without any knowledge of where the equations come from or what they represent. It is our belief that we can take advantage of information about the structure of our knowledge representation and incorporate this knowledge into the equations to reduce the size and complexity of the problems that are fed to the Simplex method thereby reducing the amount of time necessary to solve for the optimum or most probable solution. To improve the computational efficiency of the inferencing algorithm we have three areas we can attack: the Simplex phase 1 calculations, the Simplex phase 2 calculations, and the branch and bound process.

3.2 Jumpstart Solution

3.2.1 Initial Jumpstart. It was observed in early, very preliminary jumpstart work [32, 34], that phase 1 of the algorithm took an average of 60% of the total solve time during

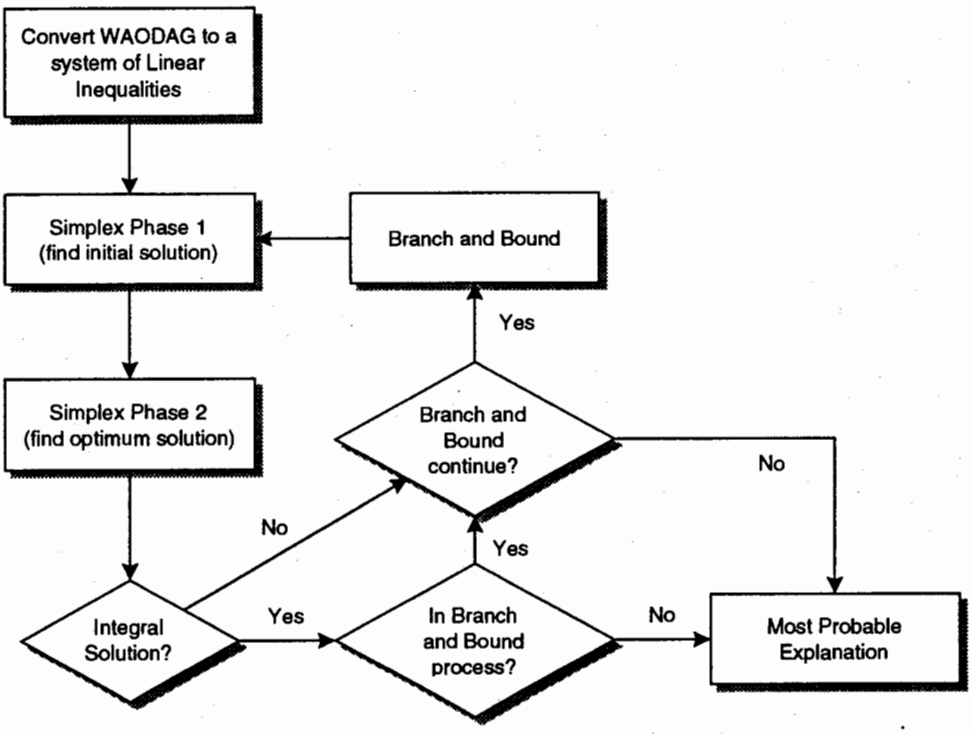


Figure 5. Process flow for solving a WAODAG.

test runs. Therefore a quick means of giving the algorithm an initial solution, or jumpstart, would allow it to bypass phase 1 (see Figure 6) and possibly decrease the time required to solve for the optimal solution. This initial solution does not need to be the best solution, only one of the possible solutions. It gets the algorithm into the feasible space of solutions from which the optimum can then be calculated.

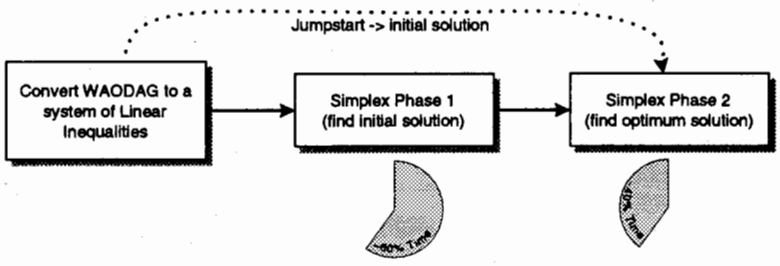


Figure 6. Jumpstart process flow.

In order to provide an initial solution we must have an assignment of truth values to the nodes in the graph representing the path of a possible solution. From equations 7 through 10 and the dependency definitions of the node types AND and OR which define the framework of the graph, we can determine how the truth values will propagate from the nodes that have been clamped or set to a value of true. Figure 7 describes this propagation.

There are two types of propagation, forward (from a node to its parents) and backward (from a node to its children). We will first consider backward propagation since that is what leads us from the evidence to our hypothesis or explanation of the evidence. If an AND node is clamped to true then all of its children must also be true. If even one of them is false, then the AND node can no longer be said to be true. If an OR node is clamped to true then we know that one or more of its children must be true. For this first phase of the jumpstart we simply pick a single node at random to clamp to true and further propagate from there. As we propagate our truth values toward the hypothesis nodes there can also be propagation back toward the evidence nodes (forward chaining or propagation). If a node is clamped to true

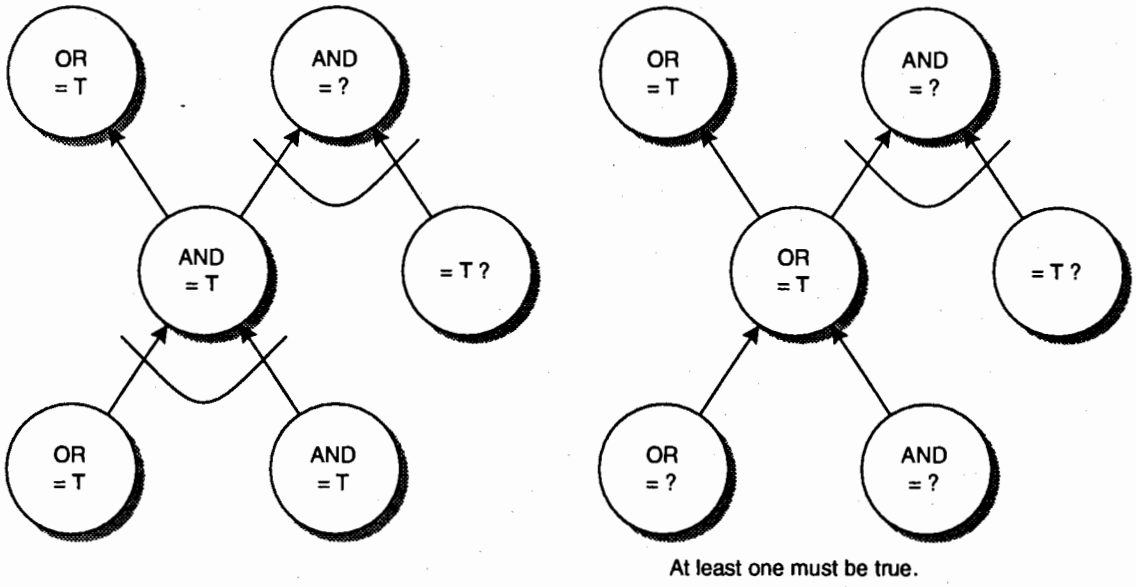


Figure 7. Truth value propagation from the AND and OR nodes.

then any of its parent nodes that are OR nodes, by direct dependency, must also be true. Any parent of the clamped node that is an AND node will also need to be clamped if all of its other children are already true and this last child completes its dependency.

Figure 8 is an example of the propagation from an evidence node B which has been clamped to true. Node B is an OR node and since it is clamped to true, we know that one or more of its children must also be true. If we pick node F and clamp it, we can further propagate the truth values. If node F is true then because it is an AND node, nodes H and I must also be true. If we then turn our attention to node F's parents we will see that node C becomes true by direct dependency. Additionally if all of node D's other children are already true then clamping F to true will cause node D to become true. We also show a continuing upward propagation from node C to node A which is a type OR parent of C.

Using the propagation information we can determine a path from the evidence nodes to the hypothesis nodes including all other nodes which must be clamped to true in between. Once we have a possible solution, the equations that represent the graph can be modified to

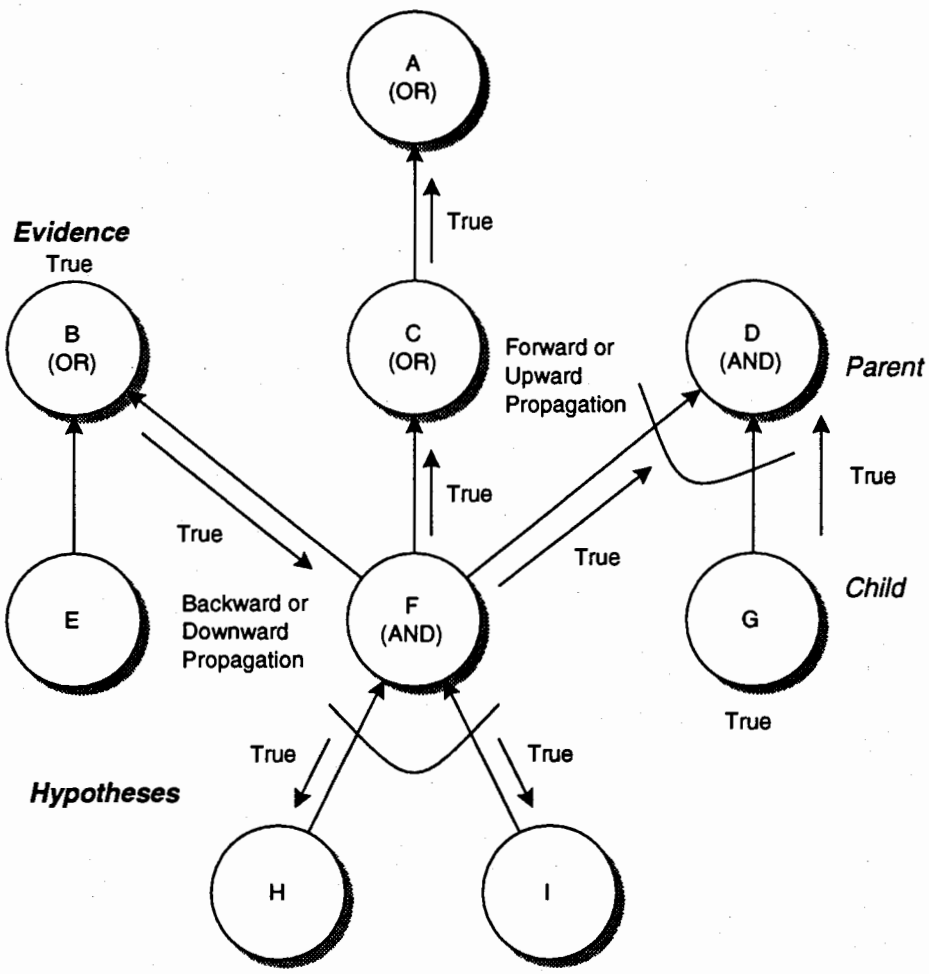


Figure 8. Truth value propagation in a small sample graph.

incorporate the solution. When the new set of equations is passed to the Simplex algorithm, it will drop immediately into phase 2 of the calculations, bypassing phase 1.

3.2.2 Incorporating the Jumpstart Solution. The method for incorporating the jumpstart solution into the equations describing the graph is straightforward and adds very little overhead to the creation of the linear program. As we create the equations representing the WAODAG we check each node to see if it is in the jumpstart solution. If the node is in the solution we negate it and subtract 1 from the opposite side of the equation [22, 32]. Any leaf nodes in the jumpstart are also negated in the cost function [22, 32]. For example in the equations

$$A \geq B$$
$$A \leq B + C$$

if A were in the jumpstart solution the equations would become

$$-A \geq B - 1$$
$$-A \leq B + C - 1$$

and if C were also in the jumpstart then the equations become

$$-A \geq B - 1$$
$$-A \leq B - C$$

Looking back at our earlier example WAODAG in Figure 4, Figure 9 shows how the equations are modified to incorporate a jumpstart solution. These modifications to the equations describing the WAODAG represent how known information is incorporated for the Simplex

Original Equations	Equations if node B is our evidence	Equations modified with Jumpstart Jumpstart: A, D, F, H
$A \geq C$	$A \geq C$	$-A \geq C-1$
$A \geq D$	$A \geq D$	$-A \geq -D$
$A \leq C+D$	$A \leq C+D$	$-A \leq C+ -D$
$B \leq D$	$1 \leq D$	$0 \leq -D$
$B \leq F$	$1 \leq F$	$0 \leq -F$
$B \geq D+F-1$	$2 \geq D+F$	$0 \geq -D+ -F$
$C \leq E$	$C \leq E$	$C \leq E$
$C \leq F$	$C \leq F$	$C \leq -F+1$
$C \geq E+F-1$	$C \geq E+F-1$	$C \geq E+ -F$
$D \geq G$	$D \geq G$	$-D \geq G-1$
$D \geq H$	$D \geq H$	$-D \geq -H$
$D \leq G+H$	$D \leq G+H$	$-D \leq G+ -H$

Figure 9. Incorporating the Jumpstart Solution into the equations.

method and is sufficient to allow the algorithm to bypass phase 1 and move directly into phase 2 to find the optimal solution.

3.3 Heuristic Jumpstart

The next area we can attempt to improve is the phase 2 calculations of the Simplex algorithm. The object is to determine an initial solution that is closer to the optimum solution than the previous jumpstart algorithm provided. We will still bypass phase 1 of the calculations but we should also reduce the number of phase 2 calculations necessary to optimize from the better initial solution. This will also help to avoid the problem that if the initial solution provided by the jumpstart algorithm is sufficiently more costly or further from the optimal solution in the feasible space than the solution found by the standard phase 1 calculations, then it is conceivable that the jumpstart may result in a longer solve time than the original non-jumpstart algorithm.

One way we can improve the solution provided by the jumpstart algorithm is by incorporating a heuristic to make better choices during the process. Two possible heuristics were

considered. Both methods employed a propagation of the costs from the leaves prior to the search. During the search for the jumpstart path any choices as to which path to take could then utilize the propagated costs in a greedy algorithm to chose the lowest local cost.

The first method starts with the costs at the leaves of the graph. We then propagate the costs in stages to each successive layer of parents until we reach the root nodes. A parent node is assigned a cost based on the costs of its child nodes. If a parent is an OR node then its cost is assigned to be the lowest cost of its children since this would be the least cost that could be incurred by clamping this node. If a node is an AND node then its cost is assigned to be the highest cost of its child costs since the node will have at least this cost if it is clamped. Figure 10 presents an example of this propagation. Notice that in this example the heuristic actually fails by picking the two nodes with costs 10 and 11 for a combined cost of 21 over the leaf with a cost of 18.

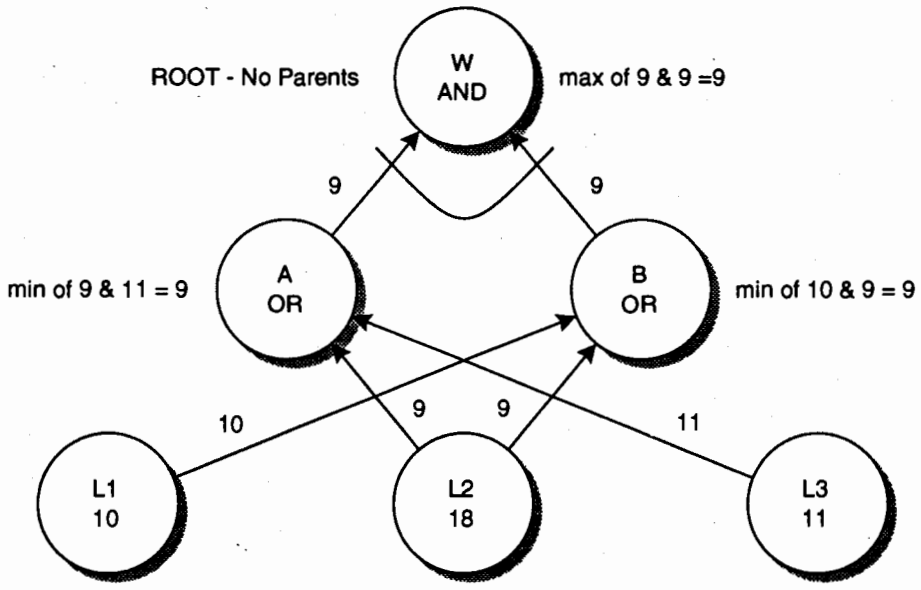


Figure 10. Cost propagation heuristic method A.

The second method also starts with the costs at the leaves and propagates costs to each successive layer of parent nodes. However, in this method nodes are assigned a cost based on

the costs of the edges entering them (edges that come from their child nodes). The process of assigning the costs is then similar to the first heuristic. If the node is an AND node then its cost is the maximum of the costs coming into it. If the node is an OR node then its cost is the minimum of the costs coming into it. The costs are propagated out of the nodes toward the parent nodes by splitting the nodes' cost equally between its outgoing edges. In Figure 11 the cost of 18 for leaf L2 is split equally between its two outgoing edges. This heuristic does not

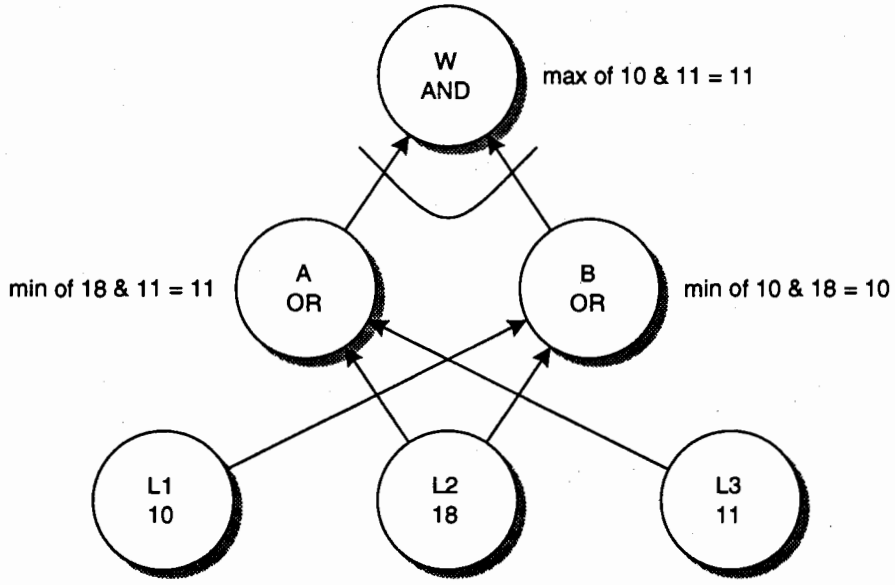


Figure 11. Cost propagation heuristic method B.

fail in the same situation as heuristic A however it also has its problems. If the cost of node L2 were 20 then node B would have to chose between two incoming edges both of cost 10 and there is the possibility that it may chose the incorrect one.

Based on studies of very similar heuristics for A^* search algorithms done in [5], the second heuristic was chosen for the implementation. The splitting of the costs in the second method seems to be more intuitive because we are sharing the cost of a node among all of its parent nodes which prevents us from propagating the total cost of the node multiple times. Also going through some small examples by hand using the first heuristic it was apparent

that it would probably not gain us much over picking a node at random. In this case either heuristic would still be an admissible heuristic since it will never prevent the optimal solution from being found and as long as we impose backtracking. The only purpose of the heuristic is to provide a better jumpstart solution and reduce the amount of time necessary to find the optimal solution in phase 2 of the calculations.

3.4 Branch and Bound

In some cases, the solution provided by the Simplex method does not correspond to a 0-1 mapping to the nodes. In this case the linear program solver algorithm employs an iterative branch and bound process [34], illustrated in Figure 12. The algorithm picks a node x_q, called the active node, that is non-integral and creates two new linear programs each with a new constraint of $x_q = 1$ or $x_q = 0$ that it in turn sends to the Simplex algorithm. Each time an integral solution is found its cost is compared to the current best solution. If it is less, then it becomes the current best. If the solution is non-integral and the total cost is less than the current best cost, a new active node is chosen from this sub-problem and two more sub-problems are created. If the total cost of the non-integral solution is greater than the current best cost or the sub-problem has no solution, then the branch is pruned. This process continues until there are no more branches to explore. At this point the current best integral solution is the overall best solution for the graph. An example of this process is shown in Figure 13.

The branch and bound process was not necessary very often, in fact early studies [34] observed that for the random WAODAGs tested only 3% required branch and bound to solve. When branch and bound is required however, the time to solve the random WAODAGs increases by an average of 236%. In addition, earlier work suggests that the percentage of real world problems requiring branch and bound to solve might be much greater than for the random WAODAGs [34]. Therefore, for completeness as well as the potential cost for requiring branch and bound in real world problems we shall attempt to improve the efficiency of the branch and bound process.

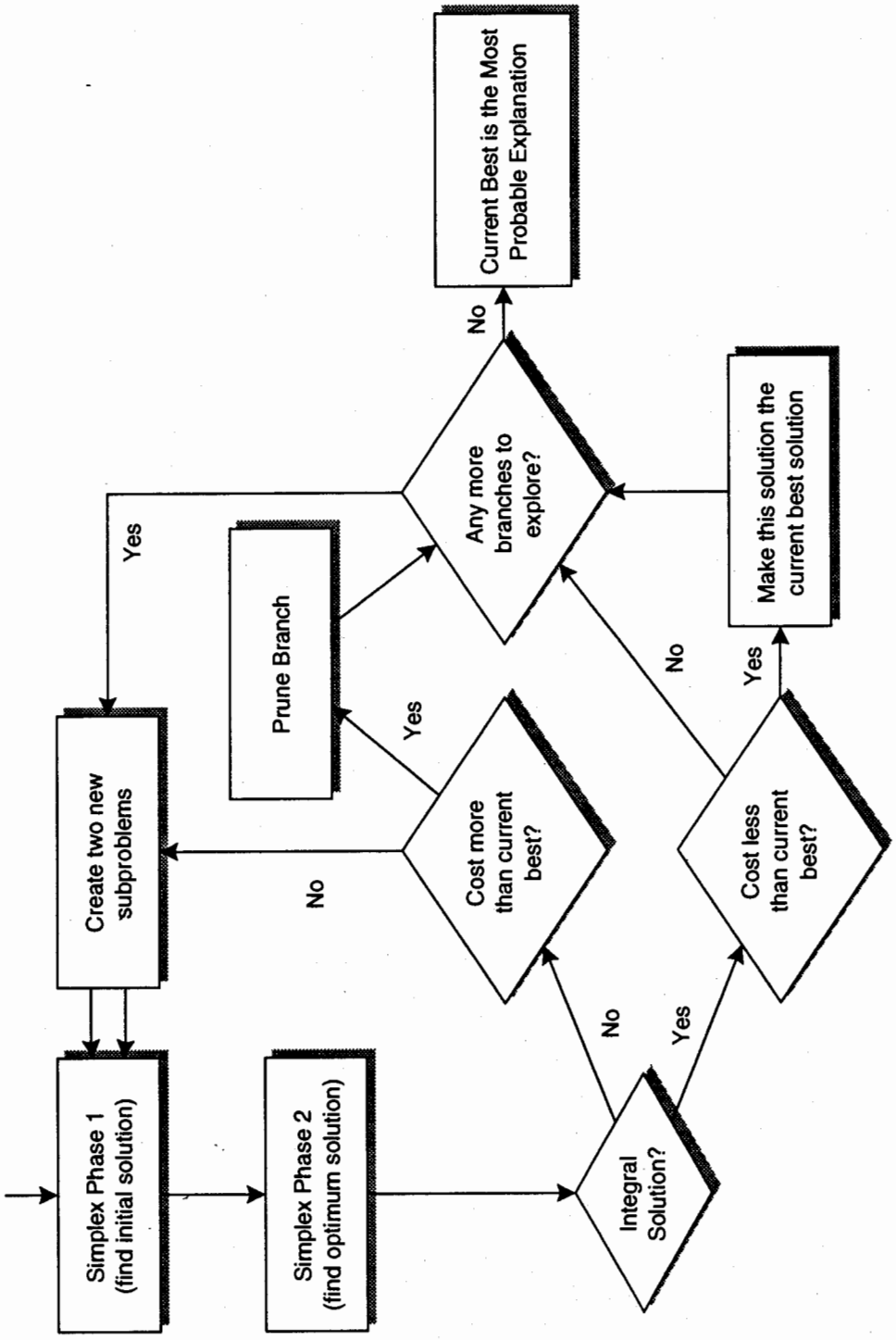


Figure 12. Branch and Bound process flow.

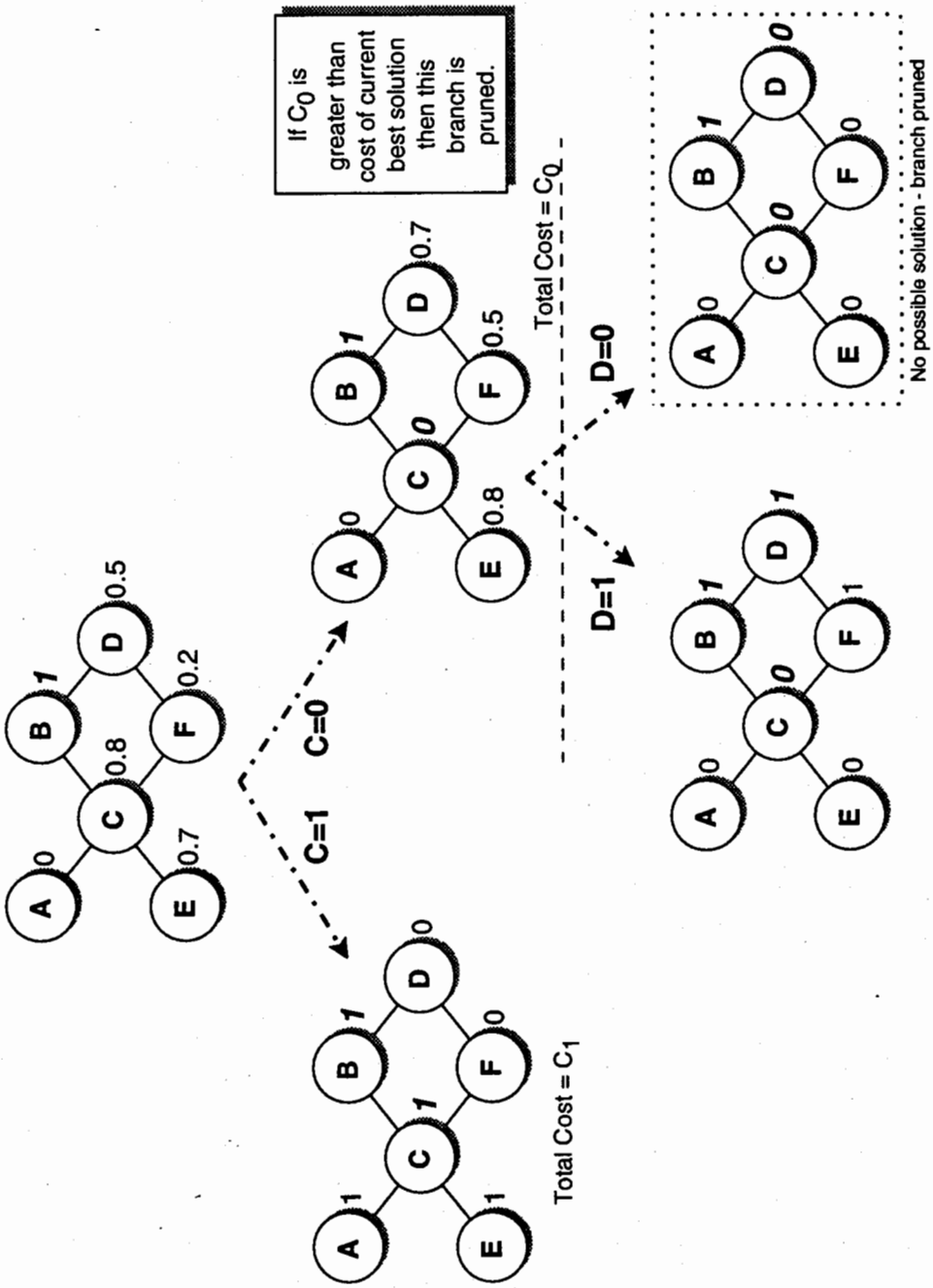


Figure 13. Branch and Bound example.

The algorithm which solves the linear programs already uses various techniques, such as the dual simplex method [22], to reduce the number of computations required to solve the multitude of linear programs created by the branch and bound process. If we could reduce the size of each of those sub-problems we should be able to substantially reduce the time to solve for the final optimal solution. The generic application created to solve arbitrary linear programs has the capability to accept additional clamping information. The application checks each node to see if clamping this node requires any additional nodes to be clamped. If a node is clamped its value is set and known so it can be eliminated from the problem. The result is that each subsequent linear program is reduced in size and should reduce the amount of time required to solve the whole problem.

3.4.1 Resultant Clamping of Other Nodes. After the equations for the boolean linear program (BLP) have been created, it is necessary to determine which other nodes must be clamped to either 1 or 0 if we clamp each node individually in the graph. Figures 14 and 15 show the local propagation of 1's and 0's from an OR node and an AND node that have been clamped to either 1 or 0.

If an OR node is clamped to 1 we can't make any determination as to which of its children are clamped. We know that at least one must be clamped but we don't know which one or ones. We can however tell that any parent nodes that are also OR nodes must additionally be clamped to 1. If there are any parent AND nodes that have all of their other children already clamped then they too must be clamped. The same clamping decisions are made for the parents of an AND node clamped to 1. For the AND node however we can also make a determination about its children; because it is an AND node, if it is clamped to 1 then all of its children must also be clamped to 1. Moving back to the OR node in the case of it being clamped to 0, we can tell that all children of the OR node must be clamped to 0 since if any were non-zero the central OR node could not be 0. Also any parent AND nodes are rendered false or clamped to 0 because of this central clamping to 0. For the AND node clamped to 0 the only clamping we can make is similar to that of the OR node with regard to any parent AND nodes.

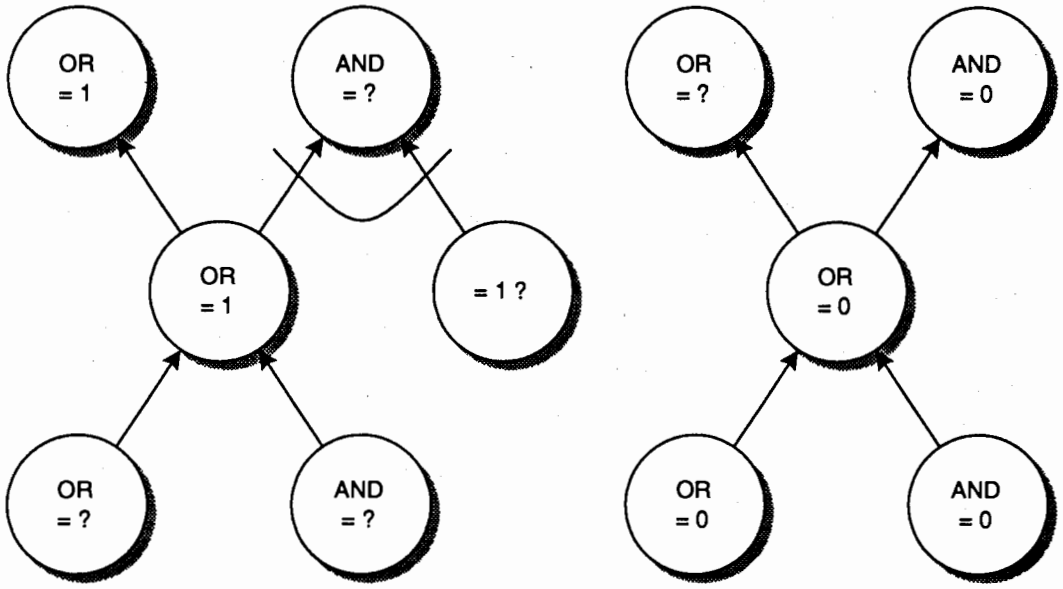


Figure 14. Local propagation of 1's and 0's from an OR node.

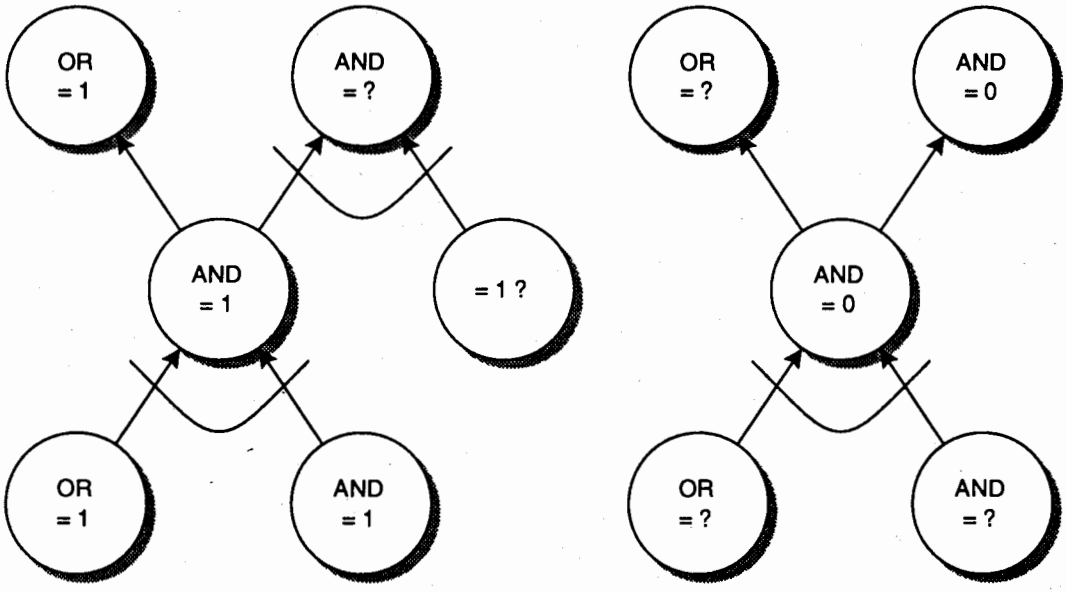


Figure 15. Local propagation of 1's and 0's from an AND node.

With this in mind, since we must already pass along the information about the active node being clamped to 0 or 1, we can also pass along the information about the resultant clampings.

3.4.2 Eliminating Unnecessary Equations. We can also use the clamping information to delete certain equations from the set describing the WAODAG to further reduce the size of the problem. The reason we can do this is because the clampings cause some of the equations to become vacuously or always true. For example if we look at equation 7 and clamp an AND node to 0 then this set of equations becomes

$$\forall x_p \in \{0,1\} \quad \{0 \leq x_p\} \tag{17}$$

which is always true, therefore this set of equations may be discarded in this particular case. Similarly if an AND node is set to 1 the equation corresponding to equation 8 may be discarded. If an OR node is set to 0 the equation corresponding to equation 9 may be discarded. Finally, if an OR node is clamped to 1 the set of equations corresponding to equation 10 may be discarded.

In addition we can add an additional equation to the system of linear inequalities utilizing the jumpstart cost since it was an integral solution. If the algorithm must resort to branch and bound it can use this cost as a lower bound to reject paths with costs greater than the jumpstart path.

With this information passed to the Simplex algorithm, should branch and bound be required, each of the linear programming problems to be solved will be significantly reduced in size. A similar approach has also been successfully used for Bayesian networks alone [33].

IV. Implementation

In this chapter we will first discuss the implementation of the benchmarking studies of the algorithms. We will then discuss the causes behind our research into the structure of large knowledge bases and how we went about studying them. Finally we examine the problem of full propagation of truth values versus backward propagation only.

4.1 Testing

The purpose for these experiments was to measure the improvement in efficiency for the new algorithms over the original algorithm. We wanted to use problems that would be representative of those that the algorithms might see in actual use as well as larger and more complex problems than are currently possible in order to show scalability. Because of the difficulty of Average Case Analysis we use empirical studies which should still show the necessary trends. Random test files were used in order to be as general as possible and to prevent testing to only a few or one particular application area.

It was necessary to develop a test methodology that would provide valid results in order to run benchmarking tests on the new algorithms and compare them to the old algorithm. It was determined that running a series of randomly generated test files on all algorithms side by side and measuring the CPU time each took to solve for the optimal solution would provide adequate results to determine the difference in efficiency trends. The UNIX *time* command was used to measure the CPU time used by each process. Additionally all of the tests were run on the same computer or on computers of equal configuration. The nominal configuration of the testbed was a Sun Sparcstation 20 with 64 MB of RAM and between 500 Meg and 1 Gigabyte of swap space. The algorithms themselves, including the original, were all written in C++ and compiled with the same optimization settings.

The initial sizes of the test files used were 10, 25, 50, 100, 200, 400, 500, 800, 1000, 1500, and 2000 nodes. It soon became clear that only the 200, 400, 500, 800, 1000, 1500, and 2000 node cases were of any real interest. The 10, 25, and 50 node graphs were solved

in less than a second and the 100 node graphs averaged 3 seconds. This did not provide enough variation in measurements to make an accurate comparison. Comparisons between the original algorithm and the enhanced algorithms is also difficult beyond the 400 node cases because the original algorithm is not able to solve (in a reasonable amount of time) the larger graphs beyond some level of complexity. When the problem size becomes too large or too complex the original algorithm gets stuck in phase 1 of the computations. The algorithm theoretically will eventually find the answer but it takes more than several hours to do so. Because of the large number of test files we ran against the algorithms, test runs that exceeded eight hours were terminated.

The original reasoning algorithm developed in [34] was used as the baseline for the testing. The original algorithm translated the graph into a system of linear inequalities which was then passed to the Simplex algorithm to find the best integral solution. This solution is the most probable explanation of our evidence. The system of linear inequalities produced by the original algorithm included no information about the structure of the knowledge representation. Tests consisted of comparing the solve time of the Simplex method against five other systems of equations: the system of equations with the jumpstart information incorporated, the system with the heuristic jumpstart information incorporated, the system with the branch and bound information incorporated, the system with the jumpstart and branch and bound information incorporated, and finally the system with the heuristic jumpstart and branch and bound information incorporated.

4.2 Graph or Knowledge Base Structure

While testing the original graph solver algorithm and the first jumpstart algorithm against a series of randomly generated WAODAGs, it was observed that a large percentage of the graphs required more than 90 percent of the total number of nodes to be true for a complete assignment to the knowledge base. This was the case even when the number of evidence nodes set to true was limited to 4 in the 200 and 500 node cases. In a medical diagnosis knowledge base this might be equivalent to saying that given a particular set of patient

symptoms, the patient has every disease and should have every other symptom (because of forward propagation of truth values).

The first randomly generated graphs had high connectivity, that is there was a large number of edges in the graph (averaged 10 percent of the maximum possible number of edges in a fully connected graph). Additionally a node at one level could connect to any other node in a lower level (topological ordering of nodes).

Various combinations of the height of the graph, level of connectivity and ratio between the number of AND and OR nodes were experimented with. The result was no significant decrease in the number of nodes clamped for a complete truth assignment. Figure 16 demonstrates what these graphs might look like.

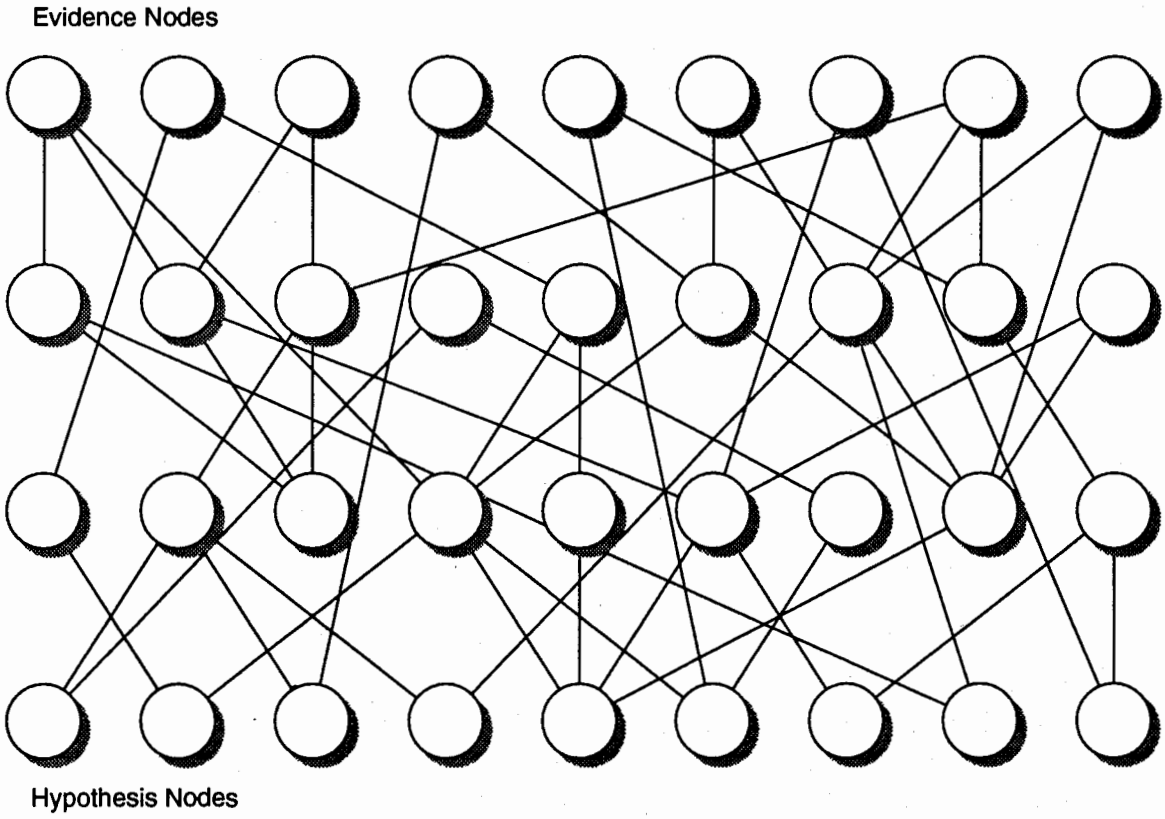


Figure 16. A purely random WAODAG.

It was therefore decided that the structure of the graph must not be typical of real world knowledge bases. A change was made to the random graph generator to add a horizontal ordering to the original vertical ordering. When the graph is generated the user inputs a range of values for the number of vertical slices in the graph. A slice metric parameter is used to control the level of cross linking between slices. The nodes in the new graph have a strong tendency (controlled by the slice metric) to connect only with the lower nodes in their own slice. This in effect creates a series of columnar subgraphs with various levels of connectivity between the subgraphs such as that shown in Figure 17.

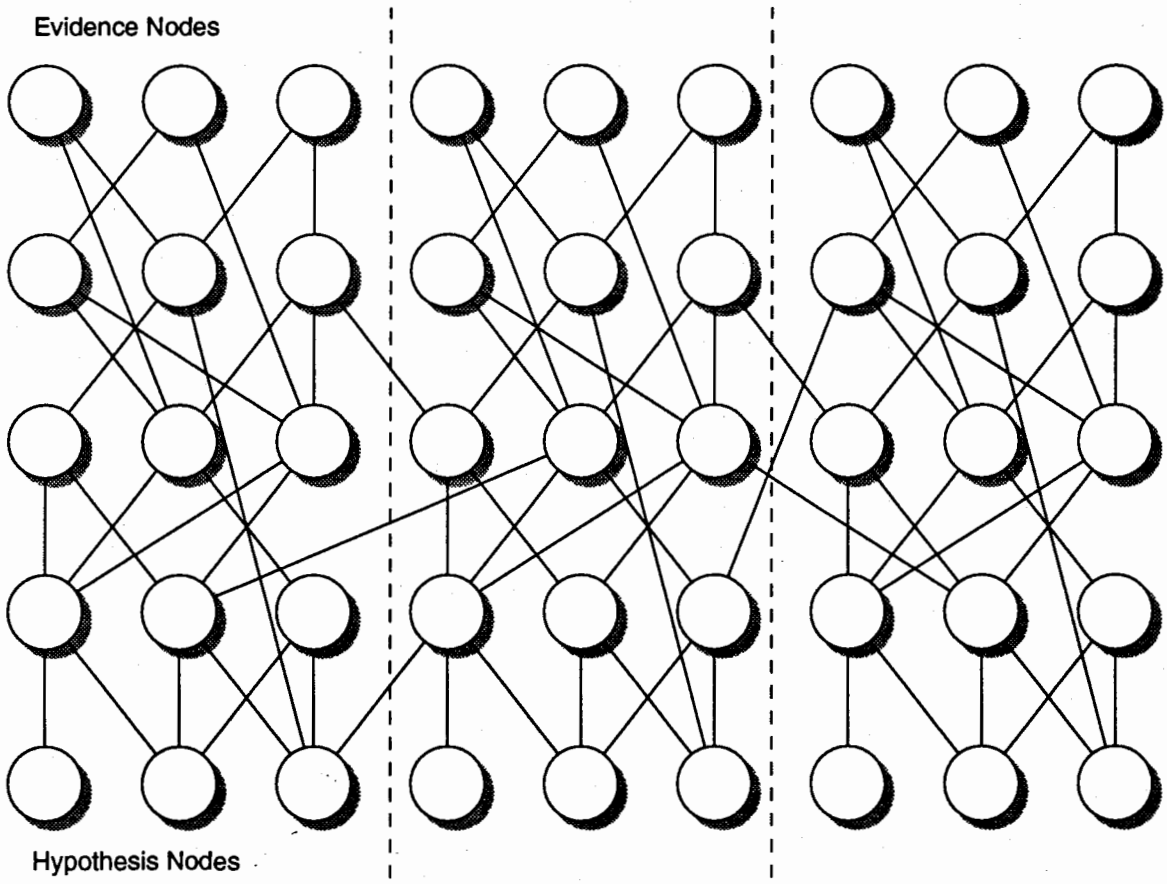


Figure 17. A columnar WAODAG.

Again, tests were run comparing the original solver algorithm to the jumpstart version of the algorithm. A number of cases were run varying the height of the graph, level of connectivity and ratio between the number of AND and OR nodes, as well as the number of vertical slices and the slice metric. In most of the cases the complete truth assignment still required more than half of the total number of nodes to be clamped to true.

4.3 Full Propagation versus Backward Propagation Only

One possible cause for this is forward or upward propagation of truth values. When we clamp a node to true, we propagate the truths backward from the evidence nodes to the cause or hypothesis nodes. If the newly clamped node is an OR type node then one of its children must be clamped to true. If the newly clamped node is an AND type node then for it to be true, all of its children must be true. Once this clamping is accomplished we forward propagate the truth values from the nodes we just clamped toward the evidence nodes. When fully propagated this gives a complete truth assignment to all nodes.

A question arises as to whether it is necessary to forward propagate truth values. In a medical diagnosis system a certain amount of forward propagation may be useful in order to let the doctor know what other symptoms should be present if the patient has been diagnosed with a particular disease.

In [6] it was shown that using backward propagation only does result in a valid solution to the problem. In fact, because it is not a complete assignment of the graph, the backward propagation only solutions should have lower costs than the full propagation solutions. The problem arises in the explanation of the solution. A minimal cost solution with a full assignment of values in the WAODAG corresponds directly to the most probable explanation in the Bayesian belief network [6], however a partial solution is missing some node assignments so the final cost does not correspond to a valid probability for that world. We cannot in fact determine what the actual probability of this world is, we only know that it is the most probable.

Therefore if we require the actual probability of a solution then full propagation, both forward and backward, is necessary. However if our only concern is what is the most probable solution then forward propagation is not necessary and backward only propagation will provide a valid solution.

V. Test Results

The primary purpose for these experiments was to compare the efficiency of the new inferencing algorithms to the original algorithm. Figure 18 gives a brief summary of the test graphs used. It is interesting to note that in previous research the largest graphs tested had at

	Nodes	Edges	Hypotheses
Minimum	100	149	13
Maximum	2000	5236	352

Figure 18. This is a summary of the test graph parameters.

most 387 nodes and a maximum of 699 edges [32].

A full summary of the test graph statistics is shown in Appendix A. It is broken down by the number of nodes and then into sub-categories to summarize the total set of test files, those that did not require branch and bound to solve, and those that did require branch and bound to solve. The total number of graphs used in the comparison was 185. Percent improvement was calculated using the accepted method [20].

It should be noted that there exists a certain inherent amount of inefficiency in the way the graphs are stored. The current method used to store the WAODAGs uses an adjacency matrix to represent the connections or edges between the nodes. In order to determine the children or parents of a node using the pre-existing software, the entire list of nodes must be searched for connections. A more efficient method of storage might be to use a linked list for the children and another for the parents of each node.

The results of the tests performed were excellent. Using the CPU time each algorithm required to solve for the optimum solution as the metric, Figure 19 shows a significant improvement over the original algorithm. The upper graph represents tests done using both forward and backward propagation, while the lower graph represents tests performed using backward propagation only. It is interesting to note that the increase in efficiency is much

more significant when considering backward propagation only. This is most likely the result of the large reduction in the number of equations necessary to describe a graph using backward propagation only. Backward propagation only actually requires only half of the equations necessary for describing full propagation. Figure 20 shows the execution time growth rate of the original reasoning algorithm compared to that of the jumpstart algorithm. The least squares best fit equations for both algorithms are presented below each graph.

From the test results we make the following observations. The largest improvement appears to come from the incorporation of the jumpstart information. The heuristic jumpstart does not actually add much in terms of a time savings over the original jumpstart algorithm. Due to the fact that so few of the random graphs had to resort to branch and bound to solve, it is difficult to estimate what effect our improvements would generally have on actual real world graphs. From the test runs we do have it seems there should be a significant time savings particularly as the graph size becomes large. Finally, the overhead of finding the jumpstart initial solution, heuristic jumpstart solution and the branch and bound information averages about 3

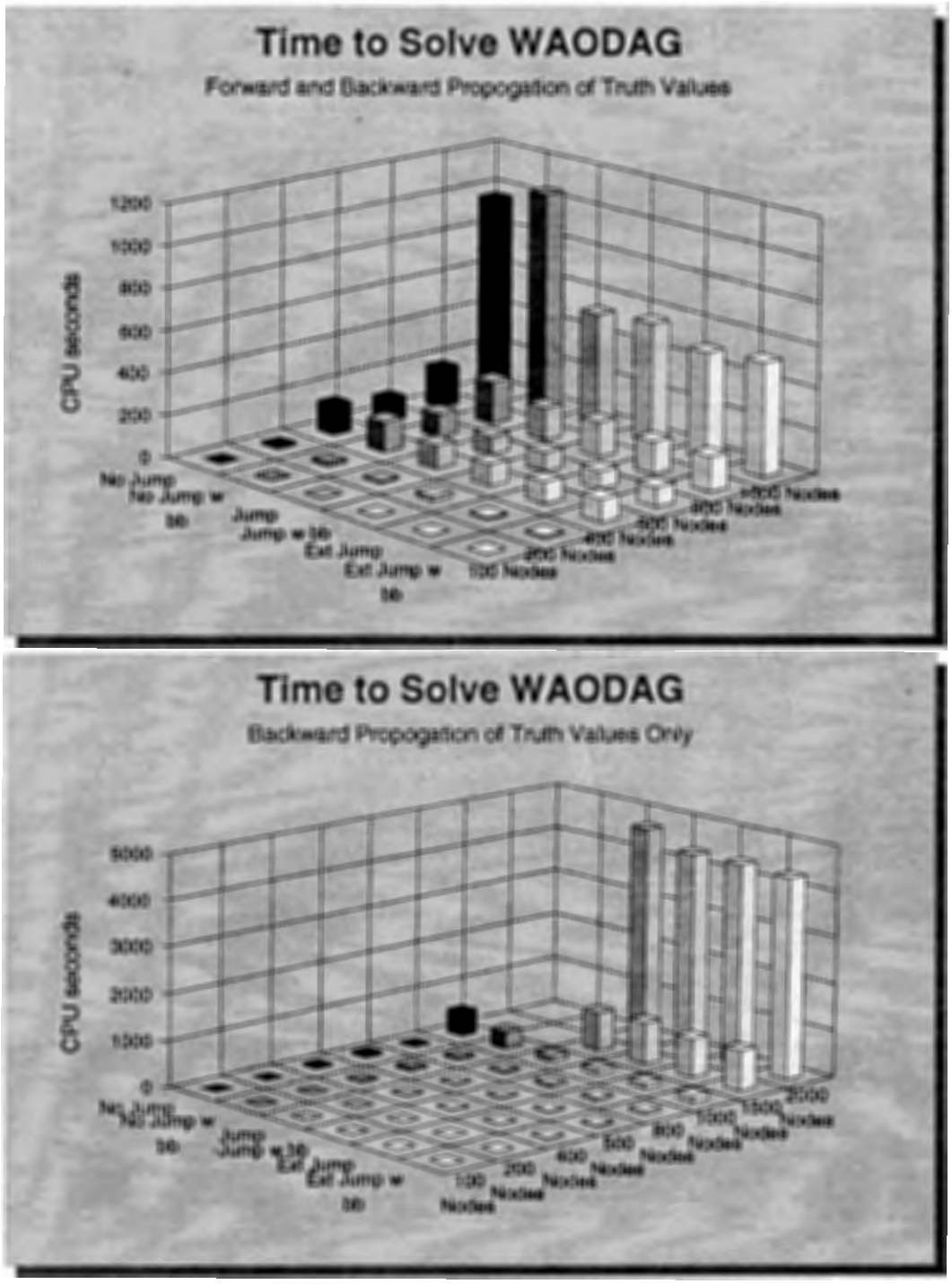
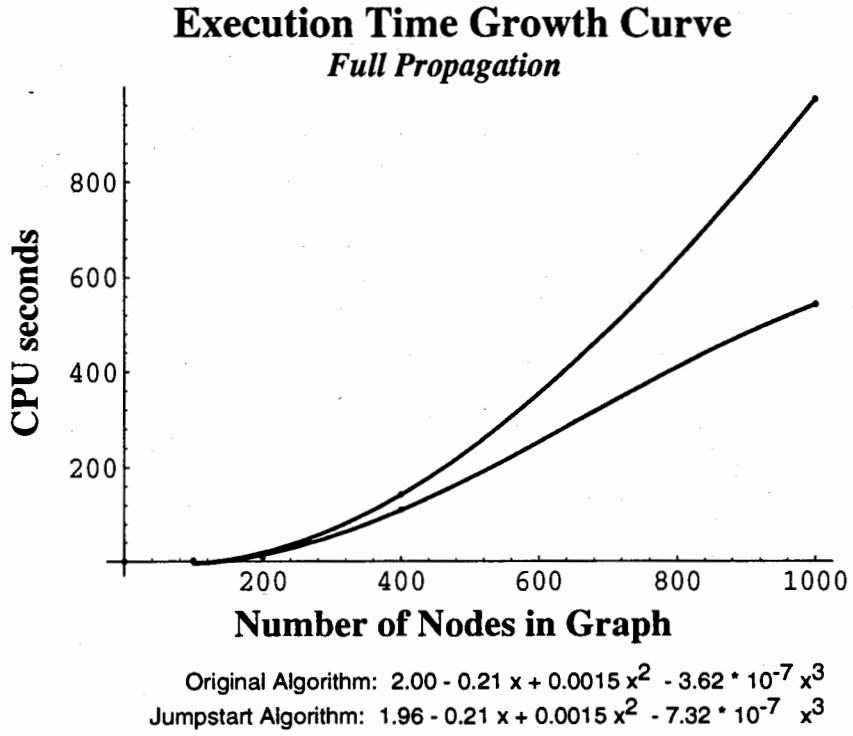


Figure 19. Results of algorithm comparison using CPU time used as the metric.



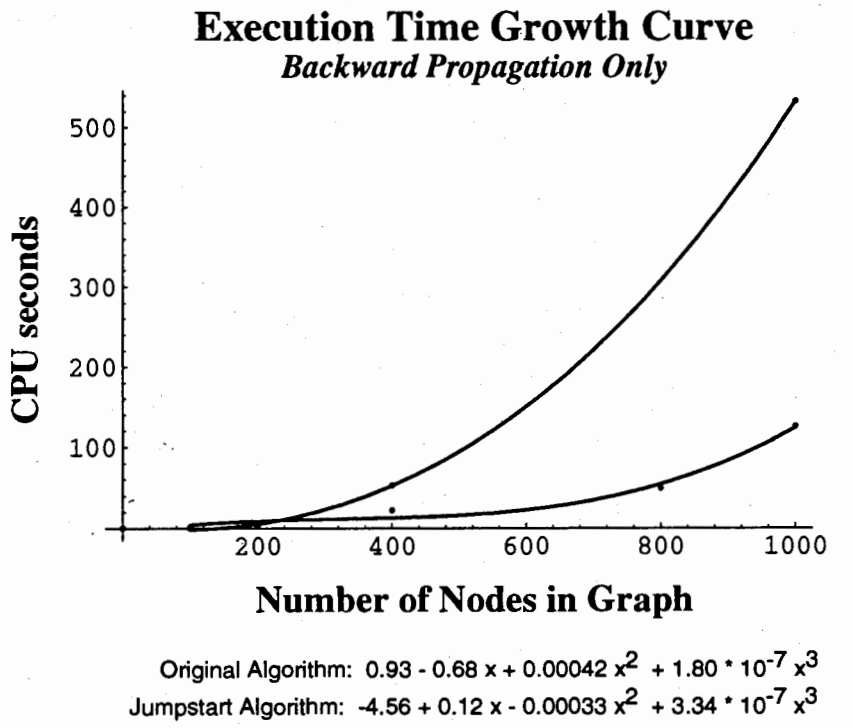


Figure 20. Execution time growth curves comparing the original algorithm and the jumpstart algorithm.

VI. Advanced Knowledge Representation - Bayesian Forest

In this chapter we will explore a new knowledge representation called a Bayesian Forest which promises to be much more versatile in representing complex real world knowledge. We will further show that our techniques for inferencing as well as problem reduction from incorporating structural knowledge can be generalized to work with this new knowledge representation.

6.1 Bayesian Forests

The research presented thus far focused on the WAODAG representation, however that representation still has some deficiencies. It cannot model cyclic information and we cannot assign costs to false states of a node. A Bayesian Forest is a new representation that subsumes both WAODAGs and Bayesian networks.

In the Bayesian Forest, the nodes of the graph represent individual instantiations of the random variables rather than the random variables themselves. This has the direct advantage of allowing us to assign probabilities/costs to all of the instantiations of a random variable. For instance we can now assign a probability/cost to a random variable A being true as well as a probability/cost for it being false. However instead of each instantiation itself being true or false they are either active or inactive and the node itself might represent a true instantiation or a false instantiation. For example if we have a boolean random variable A, its instantiations are $A = True$ and $A = False$. Random variables may also have more than two states (eg. $A = Red$, $A = Green$, $A = Blue$, and $A = Yellow$). It should also be obvious that only one instantiation of a variable can be active at any given time. In the previous example either $A = True$ is active and $A = False$ is inactive or $A = True$ is inactive and $A = False$ is active. We can create this constraint with a single equation for each node. If we let x_n denote a variable representing the state of a proposition n (eg. $x_1 \rightarrow A$) then we can let x_{n_i} denote a particular instantiation of x_n (eg. $x_{1_1} \rightarrow (A = True)$ and $x_{1_2} \rightarrow (A = False)$). If x_{n_i} is active, $x_{n_i} = 1$ and if x_{n_i} is inactive $x_{n_i} = 0$. Further, if we let I_n represent the set of all

possible instantiations of x_n then the equation is

$$\forall n \quad \sum_{\forall i \in I_n} x_{n_i} = 1 \tag{18}$$

Another significant advantage of the separate instantiations is that we can now directly represent the following type of cycle. Given random variables A, B, C, and D:

$$\{B = T\} \wedge \{C = T\} \rightarrow \{A = T\} \tag{19}$$

$$\{A = F\} \wedge \{D = T\} \rightarrow \{C = F\} \tag{20}$$

$$\tag{21}$$

which cannot be modeled by either the WAODAG or the Bayes net because this would amount to

$$B \wedge C \rightarrow A$$
$$A \wedge D \rightarrow C$$

6.2 *Converting a Bayesian Belief Network into a Bayesian Forest*

One of the great strengths of this representation is that we can take a Bayesian belief network and convert it directly into a Bayesian Forest with no loss of information. In fact the Bayesian Forest representation becomes more versatile as we shall see. If we start with the small example of a Bayesian network in Figure 21 the process to convert it to a Bayesian Forest is relatively straightforward. Each node in the Bayesian network is first separated into its individual instantiations. From the probability table we then add support nodes, one for each entry in the conditional probability table. These support nodes act as AND nodes and the instantiations of the random variables act much like OR nodes. The probabilities are assigned to the support nodes. One obvious constraint on the support nodes is that they may

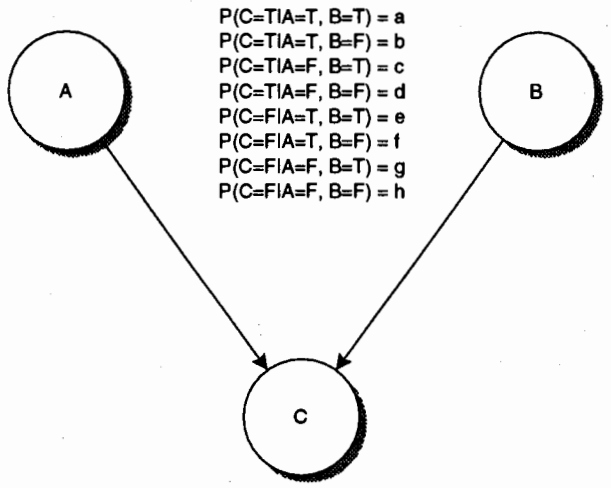


Figure 21. A simple Bayesian Belief Network.

only have a connection to one instantiation of a given random variable. Figure 22 shows the complete transformation of Figure 21 into a Bayesian Forest, assuming that we have a complete conditional probability table for the Bayesian network. The advantage of the Bayesian Forest is that complete probability tables are not required. If some of the probabilities are missing or cannot be determined the appropriate nodes are simply deleted from the graph and inferencing continues with the probabilities we do have. To find the probability of a given solution we simply multiply together the probabilities of the active nodes and during inferencing we attempt to find the assignment of nodes that yields the highest probability. Another method is to transform the probabilities into costs by taking their logarithm. As we did for WAODAGs the sum of the costs is equivalent to multiplying the probabilities. The solution with the lowest cost is then equivalent to that with the highest probability [6].

6.3 Converting a Bayesian Forest into a System of Linear Inequalities

As we can see from Figure 22 this representation is very similar to a WAODAG in this particular case. In order to use the earlier work presented in this paper to create an efficient inferencing algorithm for Bayesian Forests it is only necessary to generalize the method to

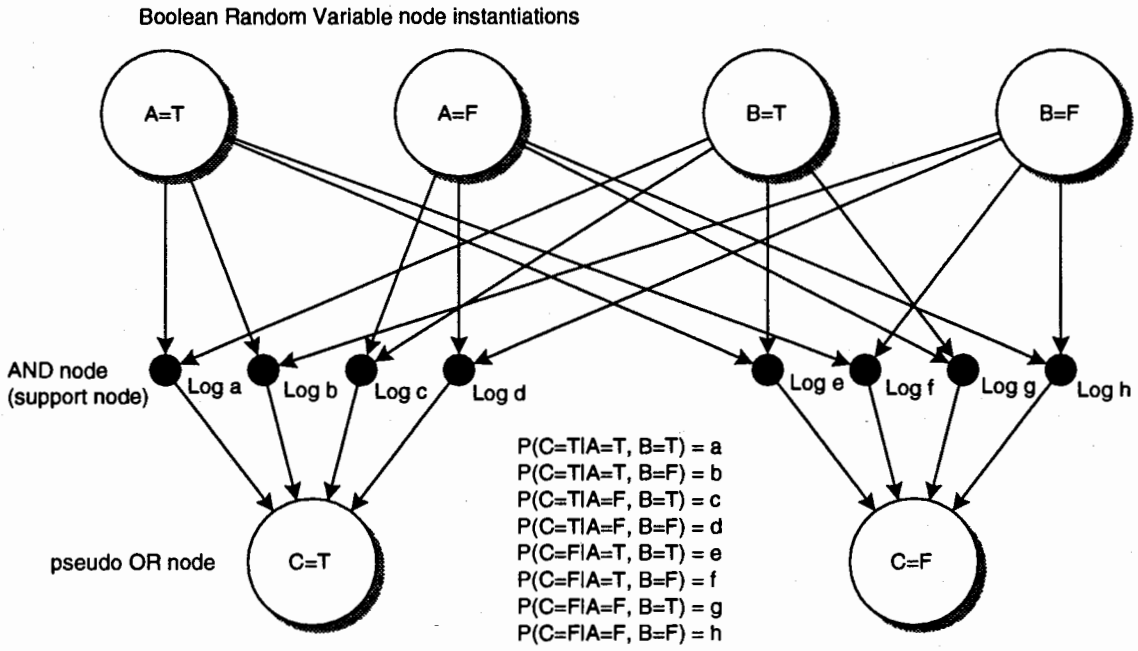


Figure 22. Converting a Bayesian network to a Bayesian Forest.

work with Bayesian Forests. Given that

$$D_n \text{ is the set of all nodes that are children of node } n \tag{22}$$

$$S \text{ is the set of all evidence nodes} \tag{23}$$

$$L_{n_i} \text{ is the set of all leaf node instantiations} \tag{24}$$

$$I_n \text{ is the set of all instantiations of node } n \tag{25}$$

$$x_n \text{ represents a support node } n \tag{26}$$

$$x_{n_i} \text{ represents a particular instantiation of some node } n \tag{27}$$

if we only allow cycles of the type shown in Equation 19 the equations for the Bayesian Forest become

$$\text{if q is an AND support node: } \forall p \in D_q \quad \{x_q \leq x_p\} \tag{28}$$

$$\text{if q is an AND support node: } \sum_{\forall p \in D_q} x_p - \mid D_q \mid +1 \leq x_q \tag{29}$$

$$\text{if } q_i \text{ is an OR node instantiation: } \sum_{\forall p \in D_q} x_p \geq x_{q_i} \tag{30}$$

$$\text{if } q_i \text{ is an OR node: } \forall p \in D_q \quad \{x_{q_i} \geq x_p\} \tag{31}$$

$$\text{Evidence: } \forall q_i \in S \quad x_{q_i} = 1 \tag{32}$$

$$\text{Cost function: } \sum_{q_i \in L_{q_i}} cost_{x_{q_i}} \times x_{q_i} \tag{33}$$

The semantics of the equations are the same as for the WAODAG [34]. Finally we add the requirement that only one instantiation of a random variable can be active at any given time.

$$\forall q \quad \sum_{\forall i \in I_q} x_{q_i} = 1 \tag{34}$$

Note that when we are creating the graph structure we must also ensure that each support node is connected to no more than one instantiation of a given node.

From the transformation in Figure 22, we also begin to see why being able to work with very large networks is important. Converting a Bayesian net to a Bayesian Forest results in a much larger graph. If there are n nodes in the Bayesian net and there are y possible states for each node then the Bayesian Forest can have as many as $y \times n$ instantiations plus a support node for every probability available.

This representation coupled with the method of converting the graph to a system of linear inequalities and solving those equations to find the most probable solution provides the necessary components for a versatile and efficient inference engine. Such a system would allow us to solve problems more efficiently as well as model and solve many more of the complex real world problems we were unable to solve previously.

VII. Conclusion

7.1 Efficient Graph Solving Algorithms

Our method of incorporating structural information from the knowledge representation into the problem prior to presenting it to the inferencing mechanism has been shown to significantly reduce the total time to solve in the random graphs tested. This method of problem reduction improves efficiency by reducing the amount of computations necessary to solve for the optimal solution. In fact the greater the reduction in problem size, the larger the initial problem to be solved can be.

Our research has further shown that if we are not concerned with the exact probability of a solution and only wish to identify that it is the best solution, then using backward propagation only allows the graphs to be solved much more quickly due to the reduced problem size. However, if we require the probability of the solution then both forward and backward propagation are necessary. A second enticement to using backward propagation only is the observation that we can solve problems at least twice as large as using full propagation.

7.2 Large Graph Structure

The empirical studies of large random graph structures have indicated some significant trends that can be related to large real world knowledge bases. The first is that very large knowledge bases do not seem to have uniformly distributed connections between nodes as shown previously in Figure 16. In fact the nodes in a large knowledge base are likely grouped together into smaller loosely connected subgroups or cells with some small amount of interaction between them. During testing the graphs which produced the most intuitive results were wide, low graphs with many columns. These graphs approached this idea of a cellular structure. The visualization of the potential large graph structure is shown in Figure 23. The concept is also supported by recent research in the field of cognitive sciences which suggests that much of our knowledge is compartmentalized.

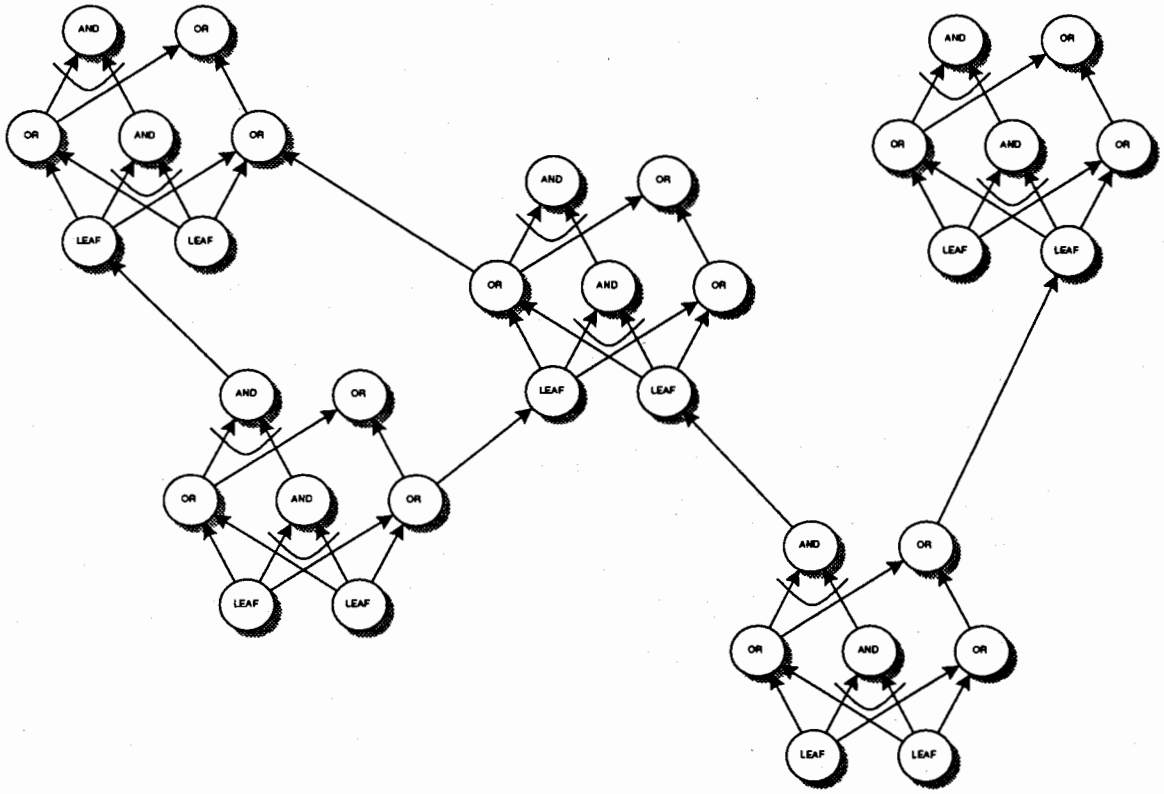


Figure 23. What most large real world knowledge base structures probably look like.

The second trend of interest is in the expected maximum depth of the graph (maximum levels of causality). Through observation of current knowledge bases [19] and our empirical studies, it is expected that there will be some maximum depth to the cells in the knowledge base and that even if the number of propositions or nodes in the graph or subgraph should double, the depth will not increase significantly, if at all. The current maximum depth of knowledge bases in use seems to be about 4 to 5 levels and even expecting our algorithms to allow solving much larger and more complex problems, the depth of the subgraphs will likely not exceed 8 levels.

7.3 The New Bayesian Forest Representation

The Bayesian Forest representation examined earlier solves several of the outstanding deficiencies with the WAODAG representation, particularly in the area of representing cyclical knowledge, as well as being able to assign costs or weights to all instantiations of a random variable instead of just the true instantiation. The Bayesian Forest has the additional advantages of allowing the conversion from a Bayesian belief network or WAODAG with no loss of information; allowing the use of random variables with more than two possible instantiations; and finally allowing reasoning with incomplete conditional probability tables. Because of its similarity to the WAODAG in structure, generalization of the techniques for inferencing as well as problem reduction from incorporating structural knowledge is relatively straightforward. This combination of the Bayesian Forest knowledge representation and the Linear Constraint Satisfaction approach for inferencing yields a powerful method of creating and efficiently reasoning with large and complex real world knowledge bases in order to solve significant problems and create useful tools.

7.4 Recommendations for Future Work

The next step is the actual implementation and testing of the Bayesian Forest representation in conjunction with the linear constraint satisfaction method for inferencing and the incorporation of structural knowledge to reduce problem size. This combination of knowledge

representation and inferencing mechanism should allow for the creation of larger and more complex expert systems to solve real world problems than have ever been possible before. Due to our results suggesting that real world knowledge bases may be loosely connected I would recommend that any implementation of the Bayesian Forest experiment with using linked lists to store the parents and children of each node rather than an adjacency matrix. This will not only speed the search for children and parent nodes but also substantially reduce the storage requirements for large graphs.

To further improve the efficiency and scalability of the algorithms further study of large knowledge base structure may provide important heuristics for subdividing problems into smaller pieces. It may be possible to precategorize or predefine the cells. Then, while the knowledge base is being created, information as to which cells a node belongs to, is incorporated into the nodes definition. This information might be used to eliminate some of the cells from the inferencing process, particularly during backward only propagation when the spreading of truth values from forward propagation is eliminated. The related field of graph drawing might also lead to additional heuristics by allowing the easier visualization of these large knowledge bases.

The other area that might be improved by additional or different heuristics is the propagation of costs in the heuristic jumpstart method. A better quick initial solution would further reduce the amount of work necessary from the linear program solver. One alternate heuristic is a variation of the one used in this research. In the current heuristic the cost of an AND node is assigned to be the maximum of the costs of the edges entering the node, since setting the AND node to true will incur at least this cost. However, as an alternate assignment, if the AND node is set to true then we know that all of its children must be true and might assign it a cost that is the sum of all the edges entering the node. This might prove to be a better heuristic for our application and provide more significant improvements over the random jumpstart method.

VIII. Glossary

AND node	A node that is true only if all of its children are true.
Any Time Algorithm	After some point the algorithm can be stopped at any time and it will have a valid answer. If left to continue the algorithm will improve this answer until it finishes with the optimum solution.
Backward Propagation	Also called backward chaining. The propagation of values from the evidence toward the possible hypotheses or causes for the evidence.
BLP	Boolean Linear Program
Branching Table	A table of information containing the local clamps possible for each possible state of each node in the graph
Child node	The nodes in the graph that a given node is dependent on for its state.
Domain	A narrow area of study such as the diagnosis of infectious diseases.
Edge	The connection between two nodes which indicates a relationship or dependency.
Evidence node	Nodes that are clamped to some truth value. These are the nodes or evidence that we are trying to explain by our inferencing.
Expert System	A framework consisting of a knowledge base, user interface and an inference engine.
Exponential Growth	The function describing the computational time to solve or growth rate is an exponential in n, with respect to the size of the knowledge base or problem we are trying to solve. For example, if n represents the size of the knowledge base then the time to solve for the optimum solution might be represented by $T(n) = C_1 e^{C_2 n}$ with C_1 and C_2 being some constants.
Feasible Space	The portion of the problem space that contains only the feasible assignments of the nodes. The vertices of the hypercube that describes this space correspond to integral solutions or assignments of the nodes.
Forward Propagation	Also called forward chaining. The propagation of values from the causes or hypotheses toward the evidence.
Heuristic	Involving or serving as an aid to learning, discovery, or problem-solving by experimental and esp. trial-and-error methods (heuristic techniques) (a heuristic assumption); also: of or relating to exploratory problem-solving techniques that utilize self-educating techniques (as the evaluation of feedback) to improve performance (a heuristic computer program). [1]
Hypothesis node	Nodes that represent explanations for our evidence.

Inference Engine	The portion of the expert system that manipulates the knowledge in the knowledge base.
Knowledge Base	A collection or database of knowledge within a narrow problem domain.
Knowledge Representation	How we represent the information we wish to reason with. Can be either graphical or textual. The representation has a large impact on what methods can be used to extract inferences from the information.
KBES	Knowledge Based Expert System
Leaf node	Nodes that have no children. In our representations they are typically shown at the bottom of the graph and represent the possible hypothesis to explain our evidence.
MBLP	Mixed Boolean Linear Program - BLP file that includes the branching table information
Multinet	A new graphical knowledge representation, similar to a WAODAG except that the nodes represent individual instantiations of the random variables rather than the random variables themselves.
NP Hard	Refers to a problem with a particular type of complexity. For a complete description see [15].
Parent node	A node that is dependent on the current node for its state.
Polynomial Growth	The function describing the computational time to solve or growth rate is a polynomial in n, with respect to the size of the knowledge base or problem we are trying to solve.For example, if n represents the size of the knowledge base then the time to solve for the optimum solution might be represented by $T(n) = C_0 + C_1 n^1 + C_2 n^2 + C_3 n^3 ...$ with C_0, C_1, C_2, and C_3 being some constants.
Problem Space	The space containing all possible combinations of node assignments whether correct or not.
OR node	A node that is true if one or more of its children are true.
Reasoning Model	See Knowledge Representation
Research	Studious inquiry or examination; esp: investigation or experimentation aimed at the discovery and interpretation of facts, revision of accepted theories or laws in the light of new facts, or practical application of such new or revised theories or laws. The collecting of information about a particular subject. [1]
Root node	Nodes that have no parents. In our representations they are typically shown at the top of the graph.

Rule Base	A knowledge base or collection of causal knowledge in the form of if-then rules. For example: IF event A occurs THEN event B will occur with a probability of X. This is one of the most intuitive and easiest ways for a knowledge engineer to encode knowledge for use by an expert system.
Uncertainty	The quality of not knowing the truthfulness of our information.
WAODAG	A weighted AND OR directed acyclic graph. A knowledge representation using AND and OR nodes to represent facts in a knowledge base as well as their interrelationships.

Appendix A. Summary of Test Results

			Num of Files	Percent of Total in Category	Average No Jump Solve Time	No Jump Solve Time Stnd Dev	Average Jump Solve Time	Jump Solve Time Stnd Dev	Average Ext Jump Solve Time	Ext Jump Solve Time Stnd Dev	Average No Jump w bb Solve Time	No Jump w bb Solve Time Stnd Dev	Average Jump w bb Solve Time	Jump w bb Solve Time Stnd Dev	Average Ext Jump w bb Solve Time	Ext Jump w bb Solve Time Stnd Dev
Forward and Back Prop Data (2)	All Trials	100 Nodes	30	100.00%	2.10	1.66	1.27	1.24	1.17	0.93	2.67	1.62	2.00	3.21	1.57	0.67
		200 Nodes	30	100.00%	14.03	13.54	9.23	7.67	9.30	8.66	14.73	11.81	19.73	57.09	10.83	10.83
		400 Nodes	13	100.00%	143.15	100.46	110.54	90.86	102.23	96.91	139.85	79.90	103.62	59.82	105.31	89.68
		500 Nodes	37	100.00%	106.86	99.29	82.46	73.65	82.97	74.40	111.76	99.36	89.73	78.37	90.70	79.27
		800 Nodes (1)	15	100.00%	189.87	155.38	153.40	117.04	153.20	98.95	199.80	155.69	164.47	116.40	164.13	99.20
		1000 Nodes (1)	40	100.00%	973.45	1092.80	542.95	531.57	505.25	417.99	1059.85	1324.35	572.20	540.97	533.35	436.87
	Branch and Bound not required	100 Nodes	28	93.33%	1.70	0.72	1.07	0.65	1.04	0.63	2.32	0.93	1.43	0.62	1.54	0.63
		200 Nodes	28	93.33%	10.89	6.44	8.29	5.87	7.93	4.99	12.11	6.67	9.18	5.16	9.00	5.01
		400 Nodes	10	76.92%	117.10	59.24	68.90	31.47	68.40	28.65	122.70	59.48	76.20	32.72	74.90	30.22
		500 Nodes	35	94.59%	97.00	88.72	80.34	72.27	80.80	73.16	102.54	89.76	85.74	73.18	86.80	74.69
		800 Nodes (1)	15	100.00%	189.87	155.38	153.40	117.04	153.20	98.95	199.80	155.69	164.47	116.40	164.13	99.20
		1000 Nodes (1)	39	97.50%	872.72	904.96	540.26	538.07	500.36	422.18	907.21	931.04	569.92	547.67	529.62	441.80
	Branch and Bound required (3)	100 Nodes	2	6.67%	7.50	1.50	4.00	3.00	3.00	2.00	7.50	1.50	10.00	9.00	2.00	1.00
		200 Nodes	2	6.67%	58.00	10.00	22.50	14.50	28.50	19.50	51.50	4.50	167.50	158.50	36.50	26.50
		400 Nodes	3	23.08%	230.00	149.11	249.33	86.19	215.00	146.40	197.00	107.81	195.00	32.87	206.67	135.83
		500 Nodes	2	5.41%	279.50	114.50	119.50	86.50	121.00	85.00	273.00	119.00	159.50	121.50	159.00	117.00

(1) These results do not include the tests graphs that the original algorithm could not solve. These were eliminated because they would have scewed the results too much.
(2) Using Forward and Backward Propagation, none of the algorithms could solve the 1500 and 2000 node cases in any reasonable amount of time.
(3) There were no 800 or 1000 node test graphs that required branch and bound

Figure 24. Summary Test Statistics for Forward and Backward Propagation

			Num of Files	Percent of Total in Category	Average No Jump Solve Time	No Jump Solve Time Stnd Dev	Average Jump Solve Time	Jump Solve Time Stnd Dev	Average Ext Jump Solve Time	Ext Jump Solve Time Stnd Dev	Average No Jump w bb Solve Time	No Jump w bb Solve Time Stnd Dev	Average Jump w bb Solve Time	Jump w bb Solve Time Stnd Dev	Average Ext Jump w bb Solve Time	Ext Jump w bb Solve Time Stnd Dev
Back Prop Only Data	All Trials	100 Nodes	30	100.00%	1.27	0.84	0.17	0.37	0.27	0.51	0.73	0.63	0.57	0.50	0.63	0.55
		200 Nodes	30	100.00%	3.23	3.90	2.93	4.63	2.23	1.20	3.90	3.51	3.80	4.48	3.03	1.20
		400 Nodes	14	100.00%	53.00	95.78	22.43	27.64	18.93	13.71	39.79	52.01	22.79	19.13	20.00	8.19
		500 Nodes	38	100.00%	81.53	192.27	24.55	17.78	27.11	21.38	81.68	193.09	29.45	19.21	33.39	26.41
		800 Nodes	15	100.00%	46.80	22.42	49.33	23.36	54.53	23.15	60.33	25.67	62.40	26.90	71.40	37.81
		1000 Nodes	44	100.00%	533.14	1481.76	126.09	81.09	126.00	73.05	382.05	728.52	143.64	81.98	145.14	76.47
		1500 Nodes (4)	5	100.00%	0.00		843.60	967.53	838.80	932.60	0.00		900.00	948.64	875.00	890.68
		2000 Nodes (4)	5	100.00%	0.00		4703.00	2392.15	4387.80	2156.32	0.00		4367.00	1728.93	4333.40	1806.12
	Branch and Bound not required	100 Nodes	20	66.67%	1.00	0.43	0.20	0.40	0.20	0.40	0.50	0.50	0.40	0.49	0.55	0.59
		200 Nodes	24	80.00%	2.08	1.26	3.04	5.15	2.13	1.24	2.92	1.15	3.96	4.98	2.92	1.26
		400 Nodes	8	57.14%	13.88	4.75	13.50	5.41	13.50	4.21	16.50	5.24	16.38	5.87	16.63	4.90
		500 Nodes	30	78.95%	18.40	11.28	18.53	8.95	19.47	9.35	23.50	12.41	23.27	9.75	24.60	10.23
		800 Nodes	15	100.00%	46.80	22.42	49.33	23.36	54.53	23.15	60.33	25.67	62.40	26.90	71.40	37.81
		1000 Nodes	33	75.00%	105.97	78.15	98.48	66.75	99.58	55.03	122.70	90.37	114.67	64.60	117.97	58.19
		1500 Nodes (4)	4	80.00%	0.00		405.00	456.39	418.00	449.24	0.00		491.50	539.01	486.75	487.80
		2000 Nodes (4)	5	100.00%	0.00		4703.00	2392.15	4387.80	2156.32	0.00		4367.00	1728.93	4333.40	1806.12
	Branch and Bound required (5)	100 Nodes	10	33.33%	1.40	0.92	0.10	0.30	0.40	0.66	1.20	0.60	0.90	0.30	0.80	0.40
		200 Nodes	6	20.00%	7.83	6.57	2.50	0.76	2.67	0.94	7.83	6.07	3.17	0.90	3.50	0.76
		400 Nodes	6	42.86%	105.17	128.88	34.33	38.67	26.17	17.98	70.83	67.73	31.33	26.08	24.50	9.43
		500 Nodes	8	21.05%	318.25	322.71	47.13	23.58	55.75	28.35	299.88	340.90	52.63	26.76	66.38	39.27
		1000 Nodes	11	25.00%	1814.64	2564.07	208.91	61.53	205.27	62.28	1160.09	1136.37	230.55	65.51	226.64	66.16
		1500 Nodes (4)	1	20.00%	0.00		2598.00	0.00	2522.00	0.00	0.00		2534.00	0.00	2428.00	0.00

(4) The original algorithm was not able to solve the 1500 or 2000 node graphs even when using backward propagation only.
(5) There were no 800 or 2000 node graphs in our test suite that required branch and bound to solve.

Figure 25. Summary Test Statistics for Backward Propagation Only

			Overall Improvement No Jump to Jump	Overall Improvement No Jump to Ext Jump	Overall Improvement Jump to Ext Jump	Overall Improvement No Jump to No Jump w bb	Overall Improvement No Jump to Jump w bb	Overall Improvement No Jump to Ext Jump w bb
Forward and Back Prop Data	All Trials	100 Nodes	66.06%	80.30%	8.57%	-21.12%	5.17%	34.26%
		200 Nodes	51.99%	50.90%	-0.00%	-4.75%	-28.89%	29.54%
		400 Nodes	29.51%	40.03%	10.00%	2.37%	38.16%	35.94%
		500 Nodes	29.60%	28.79%	1.57%	-4.38%	19.10%	17.82%
		800 Nodes	23.77%	23.93%	3.45%	-4.97%	15.44%	15.68%
		1000 Nodes	79.29%	92.67%	10.03%	-8.15%	70.12%	82.52%
	Branch and Bound not required	100 Nodes	59.01%	64.50%	3.45%	-26.61%	19.26%	10.94%
		200 Nodes	31.47%	37.39%	5.45%	-10.03%	18.68%	21.03%
		400 Nodes	69.96%	71.20%	3.37%	-4.56%	53.67%	56.34%
		500 Nodes	20.73%	20.05%	1.71%	-5.41%	13.13%	11.75%
		800 Nodes	23.77%	23.93%	3.45%	-4.97%	15.44%	15.68%
		1000 Nodes	61.54%	74.42%	10.62%	-3.80%	53.13%	64.78%
	Branch and Bound required	100 Nodes	87.50%	150.00%	33.33%	0.00%	-25.00%	275.00%
		200 Nodes	157.78%	103.51%	-21.05%	12.62%	-65.37%	58.90%
		400 Nodes	-7.75%	6.98%	16.77%	16.75%	17.95%	11.29%
		500 Nodes	133.89%	130.99%	0.00%	2.38%	75.24%	75.79%

Figure 26. Improvement Percentages for Forward and Backward Propagation

			Overall Improvement No Jump to Jump	Overall Improvement No Jump to Ext Jump	Overall Improvement Jump to Ext Jump	Overall Improvement No Jump to No Jump w bb	Overall Improvement No Jump to Jump w bb	Overall Improvement No Jump to Ext Jump w bb
Back Prop Only Data	All Trials	100 Nodes	660.00%	375.00%	-37.50%	72.73%	123.53%	100.00%
		200 Nodes	10.23%	44.78%	33.33%	-17.09%	-14.91%	6.59%
		400 Nodes	136.31%	180.00%	28.88%	33.21%	132.60%	165.00%
		500 Nodes	232.05%	200.78%	-3.90%	-0.19%	176.85%	144.13%
		800 Nodes	-5.14%	-14.18%	-1.37%	-22.43%	-25.00%	-34.45%
		1000 Nodes	322.82%	323.12%	9.45%	39.55%	271.17%	267.33%
		1500 Nodes	0.00%	0.00%	5.05%	0.00%	-100.00%	-100.00%
		2000 Nodes	0.00%	0.00%	7.79%	0.00%	-100.00%	-100.00%
	Branch and Bound not required	100 Nodes	400.00%	400.00%	0.00%	100.00%	150.00%	81.82%
		200 Nodes	-31.51%	-1.96%	46.00%	-28.57%	-47.37%	-28.57%
		400 Nodes	2.78%	2.78%	8.70%	-15.91%	-15.27%	-16.54%
		500 Nodes	-0.72%	-5.48%	3.72%	-21.70%	-20.92%	-25.20%
		800 Nodes	-5.14%	-14.18%	-1.37%	-22.43%	-25.00%	-34.45%
		1000 Nodes	7.60%	6.42%	10.06%	-13.63%	-7.58%	-10.17%
		1500 Nodes	0.00%	0.00%	6.75%	0.00%	-100.00%	-100.00%
		2000 Nodes	0.00%	0.00%	7.79%	0.00%	-100.00%	-100.00%
	Branch and Bound required	100 Nodes	1300.00%	250.00%	-75.00%	16.67%	55.56%	75.00%
		200 Nodes	213.33%	193.75%	-6.25%	0.00%	147.37%	123.81%
		400 Nodes	206.31%	301.91%	42.14%	48.47%	235.64%	329.25%
		500 Nodes	575.33%	470.85%	-12.83%	6.13%	504.75%	379.47%
		1000 Nodes	768.62%	784.01%	8.65%	56.42%	687.11%	700.68%
		1500 Nodes	0.00%	0.00%	4.08%	0.00%	-100.00%	-100.00%

Figure 27. Improvement Percentages for Backward Propagation Only

Test Suite			Graph Parameters												
			Num Nodes	Num Edges	Max Children	Avg Children	Max Parents	Avg Parents	Root Nodes	Clamped Nodes	Leaf Nodes	Graph Depth	Vertical Slices	Slice Metric	AND to OR node ratio
	100n	All Trials Avg	100	236.97	7.80	2.77	10.17	1.87	32.93	2.87	26.07	4.43	2.93	0.25	0.70
		No B and B Avg	100	227.70	7.50	2.60	10.05	1.80	33.75	3.00	26.70	4.30	2.70	0.25	0.70
		B and B Avg	100	255.50	8.40	3.10	10.40	2.00	31.30	2.60	24.80	4.70	3.40	0.25	0.70
	200n	All Trials Avg	200	451.90	8.23	2.43	11.00	1.63	63.30	2.63	48.13	5.63	3.00	0.25	0.70
		No B and B Avg	200	452.29	8.25	2.46	10.96	1.63	63.04	2.63	48.38	5.67	3.08	0.25	0.70
		B and B Avg	200	450.33	8.17	2.33	11.17	1.67	64.33	2.67	47.17	5.50	2.67	0.25	0.70
	400n	All Trials Avg	400	1038.93	9.36	2.71	13.79	2.14	120.07	3.00	78.86	6.86	2.86	0.25	0.70
		No B and B Avg	400	1041.88	9.50	2.75	13.25	2.13	118.13	2.88	86.25	6.25	2.75	0.25	0.70
		B and B Avg	400	1035.00	9.17	2.67	14.50	2.17	122.67	3.17	69.00	7.67	3.00	0.25	0.70
	500n	All Trials Avg	500	1182.03	8.84	2.55	13.29	1.84	155.21	2.58	110.79	7.21	4.66	0.25	0.70
		No B and B Avg	500	1089.20	8.57	2.43	12.47	1.63	161.73	2.67	120.87	6.63	4.50	0.25	0.70
		B and B Avg	500	1530.13	9.88	3.00	16.38	2.63	130.75	2.25	73.00	9.38	5.25	0.25	0.70
	800n	All Trials Avg	800	1576.20	8.60	2.07	11.73	1.33	284.80	3.87	199.40	5.87	3.00	0.25	0.70
		No B and B Avg	800	1576.20	8.60	2.07	11.73	1.33	284.80	3.87	199.40	5.87	3.00	0.25	0.70
		B and B Avg	800	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.70
	1000n	All Trials Avg	1000	2522.48	9.50	2.59	14.73	2.09	295.09	2.30	193.64	8.07	4.36	0.25	0.70
		No B and B Avg	1000	2390.39	9.15	2.48	13.82	1.97	303.82	2.33	215.21	7.48	4.97	0.25	0.70
		B and B Avg	1000	2918.73	10.55	2.91	17.45	2.45	268.91	2.18	128.91	9.82	2.55	0.25	0.70
	1500n	All Trials Avg	1500	4370.00	10.00	2.60	18.00	2.60	398.80	1.40	170.00	10.00	5.00	0.25	0.70
		No B and B Avg	1500	4210.50	10.00	2.50	17.50	2.50	411.00	1.50	168.75	10.00	5.00	0.25	0.70
		B and B Avg	1500	5008.00	10.00	3.00	20.00	3.00	350.00	1.00	175.00	10.00	5.00	0.25	0.70
	2000n	All Trials Avg	2000	5140.20	9.60	2.20	19.00	2.00	563.60	2.00	246.20	10.40	5.00	0.25	0.70
		No B and B Avg	2000	5140.20	9.60	2.20	19.00	2.00	563.60	2.00	246.20	10.40	5.00	0.25	0.70
		B and B Avg	2000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.00	0.25	0.70

Figure 28. Test Graph Summary Statistics

Bibliography

1. *Websters Ninth New Collegiate Dictionary, First Digital Edition.* Merriam Webster Inc., 1992.
2. Adlassnig, K.P. and G. Kolarz. "Cadiag-2: Computer Assisted Medical Diagnosis using Fuzzy Subsets," *Approximate Reasoning in Decision Analysis*, 219–247 (1982).
3. Buchanan, Bruce G. and Edward H. Shortliffe. *Rule-Based Expert Systems.* Addison Wesley, 1984.
4. Caudill, Maureen. "The Possibilities of Probabilities," *AI Expert*, *8*(3):28–31 (1993).
5. Charniak, Eugene and Saadia Husain. "A New Admissible Heuristic for Minimal-Cost Proofs." *Proceedings of the AAAI Conference.* 446–451. 1991.
6. Charniak, Eugene and Solomon E. Shimony. "Probabilistic Semantics for Cost Based Abduction." *Proceedings of the AAAI Conference.* 106–111. 1990.
7. Cheesman, P. "In Defense of Probability." *Proceedings of the Ninth IJCAI:.* 1985.
8. Cooper, Gregory F. *Probabilistic Inference Using Belief Networks is NP-hard.* Technical Report KSL-87-27, Medical Computer Science Group, Stanford University, 1987.
9. Cortes-Rello, E. and F. Golshani. "Uncertain Reasoning Using the Dempster-Shafer Method: an Application in Forecasting and Marketing Management," *Expert Systems*, *7*(1):9–18 (1990).
10. Dagum, Paul and Michael Luby. "Approximating Probabilistic Inference in Bayesian Belief Networks is NP-hard," *Artificial Intelligence*, *60 (1)*:141–153 (1993).
11. Dempster, A.P. "Upper and Lower Probabilities Induced by a Multivalued Mapping," *Annals of Mathematics and Statistics*, *38*(2):325–339 (1967).
12. Duda, R. O., et al. "Subjective Bayesian Methods for Rule-Based Inference Systems." *Proceedings of the National Computer Conference.* 1976.
13. E.P.D. Pednault, S.W. Zucker and L.V. Muresan. "On the Independence Assumption Underlying Subjective Bayesian Updating," *Artificial Intelligence*, *16*(2):213–222 (1981).
14. Fieschi, M. et al. "Sphinx: An Interactive System for Medical Diagnosis Aids," *Approximate Reasoning in Decision Analysis*, 269–275 (1982).
15. Garey, Michael R. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman and Company, 1979.
16. Giarratano, Joseph and Gary Riley. *Expert Systems, Principles and Programming.* PWS Publishing Company, 1994.
17. Giles, Rogin. "Semantics for Fuzzy Reasoning," *International Journal of Man-Machine Studies*, *17*(4):401–415 (1982).
18. Glymour, Clark. "Independence Assumptions and Bayesian Updating," *Artificial Intelligence*, *25*(1):95–99 (1985).

19. Gunsch, Gregg Major, USAF, "On the maximum expected depth of a real world knowledge base." Personal Communication, 1994.

20. Hennessy, John L and David A Patterson. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1990.

21. Henry E. Kyburg, Jr. "Bayesian and Non-Bayesian Evidential Updating," *Artificial Intelligence*, *31*(3):217–293 (1987).

22. Hillier, Frederick S. and Gerald J. Lieberman. *Introduction to Operations Research.* McGraw Hill Publishing Company, 1990.

23. Ishizuka, M., K.S. Fu and J.T.P. Ya. "A Rule-Based Inference with Fuzzy Sets for Structural Damage Assessment," *Approximate Reasoning in Decision Analysis*, 261–268 (1982).

24. Johnson, Rodney W. "Independence and Bayesian Updating Methods," *Artificial Intelligence*, *29*(2):217–222 (1986).

25. Lindsay, R. K., et al. *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project.* McGraw-Hill, 1980.

26. McDermott, John. "R1: A Rule-Based Configurer of Computer Systems," *Artificial Intelligence*, *19*(1) (1982).

27. McMillan, C. *Mathematical Programming.* John-Wiley & Sons, Inc., 1975.

28. Nemhauser, George L., et al., editors. *Optimization: Handbooks in Operations Research and Management Science Volume 1, 1.* North Holland, 1989.

29. Ng, Keung-Chi and Bruce Abramson. "Uncertainty Management in Expert Systems," *IEEE Expert*, *5*(2):29–48 (1990).

30. Pearl, Judea. *Probabalistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., 1988.

31. Santos, Jr., Eugene. *Computing with Bayesian Multi-Networks.* Technical Report TR93-10, Air Force Institute of Technology, 1993.

32. Santos, Jr., Eugene. "Efficient Jumpstarting of Hill-Climbing Search for the Most Probable Explanation." *Proceedings of International Congress on Computer Systems and Applied Mathematics Workshop on Constraint Processing.* 183–194. 1993.

33. Santos, Jr., Eugene. "A Fast Hill-Climbing Approach Without An Energy Function for Finding MPE." *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence.* 1993.

34. Santos, Jr., Eugene. "A Linear Constraint Satisfaction Approach to Cost-Based Abduction," *Artificial Intelligence*, *65*(1):1–28 (1994).

35. Schocken, Shimon and Paul R. Kleindorfer. "Artificial Intelligence Dialects of the Bayesian Belief Revision Language," *IEEE Transactions on Systems, Man, and Cybernetics*, *19*(5):1106–1121 (1989).

36. Schrijver, A. *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., 1986.

37. Shafer, Glen. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

38. Shafer, Glen and Prakash P. Shenoy. "Propagating Belief Functions with Local Computations," *IEEE Expert*, *1*(3):43–52 (1986).

39. Shimony, Solomon E. *Finding MAPs for Belief Networks is NP-hard*. Technical Report FC 93-09, Mathematics and Computer Science Department, Ben Gurion University, 1993.

40. Shortliffe, E. H. *Computer-Based Medical Consultation: MYCIN*. Elsevier, 1976.

41. Shortliffe, E. H. and B. G. Buchanan. "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, *23*:351–379 (1975).

42. Shortliffe, E. H. and B. G. Buchanan. "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, *23*:351–379 (1975).

43. Stallings, William. "Fuzzy Set Theory Versus Bayesian Statistics," *IEEE Transactions on Systems, Man, and Cybernetics*, *7*(3):216–219 (1977).

44. Winston, Patrick Henry. *Artificial Intelligence*. Addison-Wesley, 1992.

45. Zadeh, L.A. "Fuzzy Sets," *Information and Control*, *8*(3):338–353 (1965).

46. Zadeh, L.A. "Fuzzy Sets as a Basis for a Theory of Possibility," *Fuzzy Sets and Systems*, *1*(1):3–28 (1978).

Vita

Captain Eric P. Baenen [redacted]. He received his undergraduate degree in Electrical Engineering from Rose-Hulman Institute of Technology in June of 1989, prior to receiving a commission in the United States Air Force. Captain Baenen's first assignment was to the 6545th Test Group, Hill AFB, Utah as an Acquisition and Engineering Project Manger. From 1990 to 1991 Captain Baenen was the Engineering Annex Monitor for the Test Groups Engineering and Technical Support Services Contract, acting as the official liason between the Air Force and the Contractor's 70+ engineering personnel. In this capacity he oversaw and inspected all work done under the Engineering Annex for the Test Group. From 1991 to 1993 Captain Baenen worked on the Advanced Range Data System (ARDS) Project to revamp the Test Group's Mission Control Center to add virtual reality large screen mission display capability as well as to bring high accuracy, high dynamic Global Positioning System capability to test vehicles operating in and around the Utah Test and Training Range. Captain Baenen also helped design, managed the installation of, and was the first systems manager of the Test Group's first local area computer network spanning nine buildings, including two wireless spread spectrum radio links to connect over 250 PCs, workstations, and mainframes.

Captain Baenen can be contacted by [redacted].

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
Generalized Probabilistic Reasoning and Empirical Studies on Computational Efficiency and Scalability

5. FUNDING NUMBERS

6. AUTHOR(S)
Eric Paul Baenen, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/94D-02

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Expert Systems are tools that can be very useful for diagnostic purposes, however current methods of storing and reasoning with knowledge have significant limitations. One set of limitations involves how to store and manipulate uncertain knowledge: much of the knowledge we are dealing with has some degree of uncertainty. These limitations include lack of complete information, not being able to model cyclic information and limitations on the size and complexity of the problems to be solved. If expert systems are ever going to be able to tackle significant real world problems then these deficiencies must be corrected. This paper describes a new method of reasoning with uncertain knowledge which improves the computational efficiency as well as scalability over current methods. The cornerstone of this method involves incorporating and exploiting information about the structure of the knowledge representation to reduce the problem size and complexity. Additionally, a new knowledge representation is discussed that will further increase the capability of expert systems to model a wider variety of real world problems. Finally, benchmarking studies of the new algorithm against the old have led to insights into the graph structure of very large knowledge bases.

14. SUBJECT TERMS
Expert System, Linear Constraint Satisfaction, Efficiency, Scalability, Inferencing, WAODAG, Bayesian Forest, Knowledge Representation

15. NUMBER OF PAGES
75

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL