COMPUTING A MAXIMAL CLIQUE USING BAYESIAN BELIEF NETWORKS

By

AMITOJ S. LIKHARI

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

This thesis is dedicated to my family without whose constant and unconditional support, this would not have happened.

ACKNOWLEDGMENTS

I would like to thank Dr. Anand Rangarajan for the countless hours he spent explaining the details of the algorithm and helping me see things clearly every time I ran into trouble.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF FIGURES vii

ABSTRACT

COMPUTING A MAXIMAL CLIQUE USING BAYESIAN BELIEF NETWORKS .. viii

CHAPTER

1 INTRODUCTION 1

2 MATHEMATICAL BACKGROUND 4

Probability Theory 4
Applications of Bayesian Belief Networks 15

3 INFERENCE AND LEARNING IN BAYESIAN NETWORKS 16

Inference Algorithms 16
Sum-Product Algorithm 18
MIME Algorithm 21
Self-Information and Mutual Information 21
Self-Information 21
Mutual Information 22
Entropy 22
Conceptual Explanation of the MIME Algorithm 23
MIME: Mathematical Explanation 24
The MIME Algorithm 27
Complexity Analysis of the Algorithm 30

4 MAXIMUM CLIQUE AND OTHER NP COMPLETE PROBLEMS 32

Polynomial Time Problems 32
Some Important NP Complete Problems 34
The Maximum Clique Problem 34
Complexity 35
Problem Formulations 35
Integer Formulations 36

Edge Formulation 36
Independent Set Formulation 36
Algorithms for the Maximum Clique Problem 37
Exact Algorithms 37
Heuristics 38
Benchmark Approaches 40
Applications of the Maximum Clique Problem 40

5 IMPLEMENTATION DETAILS 42

Adapting the MIME 42
Design and Implementation 44
Design 45
Implementation 53

6 RESULTS AND CONCLUSIONS 60

Testing 60
Results 61
Discussion of Results 62
Future research 63
Applications 64

APPENDIX

RESULTS OF THE BBN ON DIMACS GRAPHS 66

LIST OF REFERENCES 68

BIOGRAPHICAL SKETCH 72

LIST OF FIGURES

Figure page

2.1 A Simple Bayesian belief network 11

3.1 MIME algorithm simplified. 29

4.1 The four classes of problems 33

5.1 UML of class Node 48

5.2 UML of class Network 51

5.3 ASCII graph format 54

5.4 Interface to the BBN 56

5.5 Format of the output file 58

6.1 Summary of results 62

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

COMPUTING A MAXIMAL CLIQUE USING BAYESIAN BELIEF NETWORKS

By
Amitoj S. Likhari

May 2003

Chair: Anand Rangarajan
Major Department: Computer and Information Science and Engineering

Bayesian belief networks (BBN) are a combination of graphical models and probability theory used for reasoning under uncertainty. These are different from the other graphical models in being directed; essentially a BBN is a directed acyclic graph that represents a probabilistic model with the nodes representing random variables. Technically, a Bayesian network describes a joint probability distribution over the set of variables. It uses conditional independence between the variables that is coded into the network structure to compute the joint probability distribution over the network. BBNs are particularly suited for solving problems in which the structure of the network is defined by a human expert or is learned from the data. This typically happens in solving real world problems like risk analysis, data mining, etc.

This thesis computes a maximal clique of a graph using BBBNs. A maximal clique is any complete subgraph of a graph. The maximum clique is the largest maximal clique. The maximum clique problem is an NP-complete problem and does not have any

optimal solution. This is the first attempt to use the BBNs to solve a combinatorial optimization problem like the maximum clique.

The thesis implements a new algorithm called the MIME (minimization of mutual information and maximization of entropy) for belief propagation (learning the probabilities) in the network. One of the advantages of this implementation of the BBNs as applied to this problem is that it does the computation on the complement graph and so performs faster on denser graphs.

CHAPTER 1
INTRODUCTION

Bayesian belief networks (BBN) are a new tool for reasoning under uncertainty. They resemble artificial neural networks (ANN) in some ways but offer some advantages: they provide the same, if not better, flexibility as neural networks; they can be trained; and they overcome the one glaring weakness of neural networks--they have a sound foundation in probability theory. As a result, they are not viewed as black-boxes for computation that "magically" come up with results like ANNs, but their working is all probabilistic and in many cases, almost logical.

All of the above have made BBNs very popular and they have been applied in ways that ANNs never were. BBNs have been applied in problems of risk assessment, stock market prediction and the other high-end problems; their most popular applications perhaps are the Microsoft Office Assistant and the Microsoft Trouble Shooter--both of which are based on BBNs.

BBNs are essentially a probabilistic model represented graphically, i.e., as a graphical model. Thus, they are a combination of the two theories: probability theory and graph theory. Representing a probabilistic model in terms of a graphical model makes the probabilistic model simpler and more logical and hence easier to understand. BBN differ from other graphical models in being directed graphical models [JORD97]. Many BBN proponents like to call this direction as a causal structure, which does not make mathematical sense because probability can be reversed and so, direction does not matter.

The direction in a BBN is used mainly to come up with a structure for the network., which is one of the three unknowns in a BBN.

There are three unknown in a BBN:

1. the structure of the network,
2. inference mechanisms for decision making and
3. the parameters of the network.

Explanation of the three unknowns is provided in detail in later chapters.

In this thesis, BBNs are applied to an NP-complete combinatorial Optimization problem--the maximum clique problem. This is the first attempt to apply BBNs to such an optimization problem.

The maximum clique problem is an NP-complete problem. Many heuristics, including NNs, have been applied to the same problem. A clique is a fully connected subgraph of a graph. Detailed definitions and explanations of the same are provided in later chapters. The thesis computes a maximal clique, i.e., a clique that is not a subset of any other clique of the graph. The maximum clique is simply the largest maximal clique of the graph.

A new belief propagation algorithm--MIME (for minimization of mutual information and maximization of marginal entropy)--was used for belief propagation in this thesis. The algorithm is guaranteed to converge to a local minimum [RANG02]. The BBN developed in the thesis was tested on graphs developed in the second DIMACS challenge to solve NP-complete problems. The BBN, in many cases, performed at par with the benchmark exact algorithm provided in the challenge. These results are highly

encouraging especially since this is the first time BBNs have been applied to an NP-complete combinatorial optimization problem.

The thesis starts with an overview of BBNs, their working and the mathematical concepts used. The process of inference, or belief propagation is then explained along with details of the MIME algorithm; that is followed by an overview of the maximum clique problem, various approaches used to solve it and so on. The design and implementation of the BBN and adapting the algorithm for the maximal clique problem is provided after that. Finally, the results obtained and the future research directions are discussed.

CHAPTER 2
MATHEMATICAL BACKGROUND

As mentioned in the previous chapter, BBNs are directed graphical models. A brief introduction to graphical models was provided. In this chapter, the same are explained in detail along with some important concepts in probability theory and explanation of the distinction between frequentist and subjectivist views of probability theory.

Probability Theory

There are two different views of probability, the more commonly known one is the *objective* or *frequentist* view, and the other one is the *subjectivist* or *Bayesian* view [HECK95]. Both views are similar in most respects except in the definition of probability. Both are concerned with probability of the occurrence of some event from a set of events. An event is the outcome of some random phenomenon or a state of some part of the world in some time interval in the past, present, or future [HECK96]. The set of all possible events is known as the sample space. The sample space of a random variable A is denoted as $S(A)$ and is the set of all the possible values that A can take, i.e., $A=a, a \in S(A)$. The probability of an event is the chance of occurrence of that event. In the *objective* or *frequentist* approach, it is the statistical chance of that event occurring. In the subjective approach, an "expert" assigns the chance of occurrence of an event. The expert could also mean the programmer. In both approaches the same rules of probability hold. The probability of occurrence of an event A is denoted as $P(A)$. A probability

distribution is a function that defines the probability of occurrence of each event in the sample space [DEAN95]. It has the following properties:

$$0 \le P(A = a) \le 1 \quad \forall x \in X \qquad \text{--(2.1)}$$

and

$$\sum_{a \in X} P(A = a) = 1. \qquad \text{--(2.2)}$$

Thus the probability value for an event ranges between 0 (absolutely no chance of the event occurring) and 1 (the event must occur). Also the sum of the probabilities over all the events that can occur in a sample space is 1, i.e., one of the events in the sample space has to occur at any point. For probability reasoning systems, this means that the possible choices of each of the variables should be exhaustive [DEAN95].

If there are two events, say *A* and *B*, that can occur at the same time, the sample space becomes a product of the sample spaces of each of the two events. This combined sample space is known as the joint sample space denoted by *S(A)* x *S(B)* or *S(A,B)*. A probability distribution governing the joint sample space that tells the state the random variables are going to be in at any point is known as the joint probability distribution over that sample space. The joint probability distribution for the same is denoted by *P(A,B)*.

In a joint sample space the probability of just one event is known as the marginal probability of the variable. The marginal probability can be computed from a joint probability distribution by summing over all the states of the other variable in the joint space; e.g. the marginal probability of *A* can be computed from the joint probability of *A* and *B* by summing over *B* [DEAN95]. Mathematically,

$$P(A) = \sum_{b} P(A, B = b) \quad \forall b \in B \qquad \text{--(2.3)}$$

This process is called marginalization and also known as the addition rule of probability. As is evident from the above rule, the joint probability distribution along with the addition rule allows the computation of the marginal probabilities of any of the variables in the joint distribution. This is an important property and is used in BBNs.

Another important concept is conditional probability. Simply put, conditional probability is the probability of an event given that another has already occurred. Thus, *P(A|B)* is the probability of *A* given the probability of *B*. Clearly this will have no effect if the events are independent of each other. For example, if *A* is the event of getting heads in a coin flip, *B* is the probability of it raining today, knowing *B* does not have any effect on the probability of *A*. If however, *B* is the probability of the coin being biased (say both sides being heads), it will definitely have an effect on the probability of *A*. The conditional probability can also be computed from the joint probability distribution as follows

$$P(A \mid B) = \frac{P(A, B)}{P(B)} \qquad \text{--(2.4)}$$

This is known as the product rule of probability. As can be seen from the above examples, the probabilities of events change depending on the observed events. One of the most commonly used methods for the manipulation of conditional probabilities and based on this notion of probabilities changing in light of new evidence is Bayes' theorem. Bayes' theorem is based on the notion that before an event occurs, there exists a probability of occurrence of that variable. Sometimes this probability, called the *prior probability* of the event because it is the probability of the event before (*prior*) the outcome of the other event is known, is computed statistically from available data; this is the frequentist approach. The data, however, are not available in all cases; in such cases

the event is assigned a probability based on the "belief" of an expert in the occurrence of the event. This is the Bayesian or subjectivist approach. In either case, the event is dependent on some other event. Knowledge of the state of that event has an effect on the probability of the first event. The probability that is computed when the outcome of the other variable is known is called the posterior probability of the variable. The mathematical formulation of Bayes' theorem is given below along with an illustration of its working with an example [DEAN95].

$$P(A_i \mid B) = \frac{P(B \mid A_i)P(A_i)}{\sum_{A_j} P(B \mid A_j)P(A_j)} \qquad \forall A,\ A_j \in A \qquad \text{--(2.5)}$$

In the above equation, $P(A_i|B)$ is the posterior probability of A_i. A_i is an event from the sample space of random variable A. The conditional probability of B given A_i and the prior probability of A_i are known. The denominator term is actually the probability of B. This is given according to the theorem of total probability [MITC97]:

$$P(B) = \sum_{i=1}^{n} P(B \mid A_i)P(A_i) \forall A_i \in A \quad \text{and} \ \sum_{i=1}^{n} P(A_i) = 1 \qquad \text{--(2.6)}$$

One of the ways the theorem is applicable to inference is to find the best hypothesis that explains the data observed. If h is the hypothesis and D is the observed data, then the posterior probability of the hypothesis in light of the data is given by

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)} \ \textit{where}\ h \in H \qquad \text{--(2.7)}$$

Listed below is an example of an application of Bayes' theorem [PEAR00]. Let the variables A, B and C represent the state of a pavement. Let A represent the probability of the pavement being wet, and B and C represent the probabilities whether the pavement is wet and there is oil on the pavement, respectively. Further, let B and C

be the only possible conditions of the pavement being slippery. Given the outcome of *A*, whether the pavement is slippery or not, and the prior probabilities of the pavement being wet or oily i.e., the probabilities of *B* and *C* respectively, it is possible to compute the posterior probability of the pavement being slippery due to being wet or oily. The prior probabilities of *B* and *C* are the probabilities of the pavement being slippery due to water or oil before it is known whether the pavement is slippery or not.

Finally, all the above can be extended to apply to set of random variables using the chain rule of probability [DEAN95]. Mathematically,

$$P(A,B,...,Z) = P(A|B,...,Z)\ P(B|C,...,Z)...P(Y|Z)P(Z). \quad \text{--(2.8)}$$

Thus, the chain rule allows a joint probability distribution to be expressed completely in terms of the conditional probabilities of the variables. However, the number of terms in the equation increases exponentially with the number of random variables. Although the number of terms in the above equation appears to increase polynomially, it should be noted that the computation is performed over all the values of the variables, i.e.,

$$P(A,...,Z) = P(A=1|B=0,..,Y=0,Z=1)\ P(A=1|B=0,..,Y=1,Z=0)...$$

$$\forall\ A, B, ..., Z.$$

This makes the total amount of space required for the same to be of the order $O(2^n)$ [MURP01a]. This space requirement can be reduced to $O(n2^k)$ if the conditional independence relationships are taken into account, where n is the maximum degree of a any node.

Conditional independence is explained as follows [MURP01a]: A is conditionally independent of B given C if

$$P(A,B|C) = P(A|C) \quad \text{--(2.9)}$$

The implications of this in Bayesian belief networks are that it eliminates listing variables that are conditionally independent of another variable thus reducing the complexity of completely representing the joint probability distribution over a set of variables. Clearly, knowing the joint probability distribution over a set of variables allows the computation of any of the marginal probabilities of any of the variables in the distribution.

Graphical models make use of the above fact and attempt to compute the marginal probabilities based on the joint probabilities. They attempt to compute the joint probability and keep it synchronized with the marginal probabilities for each variable. There are various methods to compute this. An exact computation of the probabilities and the joint probability is NP-hard [DAGU94] but heuristics are available that can give a good estimate of the probability.

Graphical models are defined to be a marriage between graph theory and probability theory [MURP01a]. A problem is represented as a directed acyclic graph (DAG) with each node representing a random variable of the problem. Therefore, there are two important parameters that need to be estimated before graphical models can be used to compute the probabilities--the structure of the graphical model and the algorithm used to compute the probabilities. The most common method of estimating the structure of the model is using the conditional independence relationships between the variables in the distribution and forming the structure based on these relationships. There are two types of graphical models based on how conditional independence is represented in the structure: undirected--two sets *A* and *B* are conditionally independent given a third set *C*

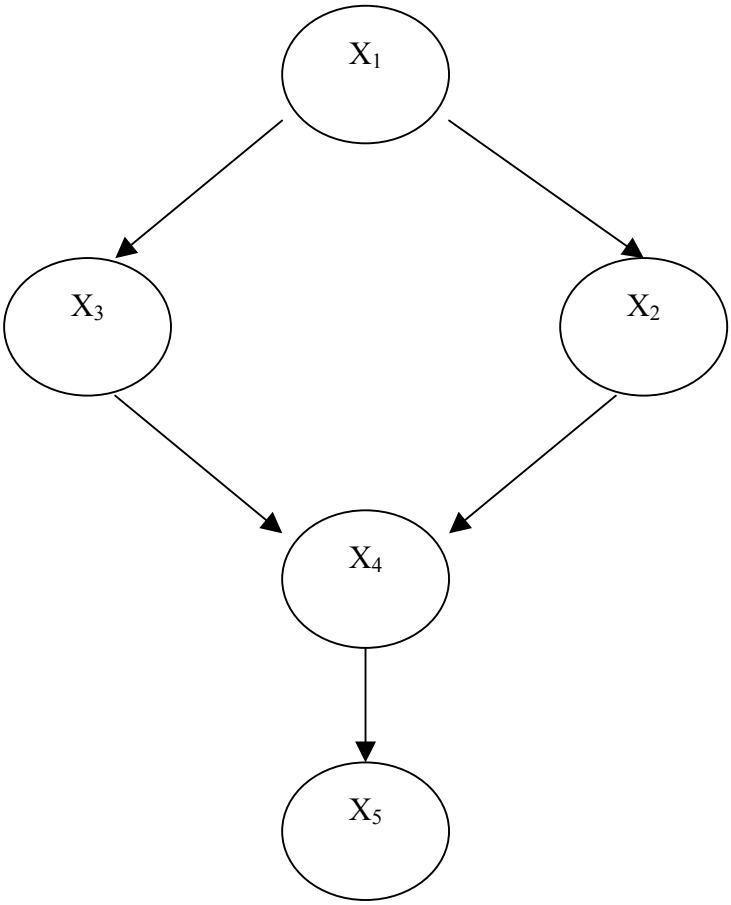
if all paths from *A* to *B* are separated by a node in *C* [MURP01a] e.g., Markov Random Fields, and directed--the absence of an arc between two nodes represents conditional independence between the variables represented by the nodes, e.g., BBNs. This is explained in detail below.

Conditional Independence in BBNs is explained as follows: A node is independent of its ancestors given its parents where the parent/child relationship is defined according to some strict topographical ordering of the nodes [MURP01a].

BBNs and their properties are explained below using an example [PEAR00]. The problem is to determine whether a pavement is slippery or not. The pavement can be slippery if and only if it is wet and for the example, there are only two ways that the pavement can be wet: either it is raining or the sprinkler is on. Further, the two causes for the pavement being wet are considered to be mutually exclusive. Finally, another variable, the season, affects the probability of both rain and the sprinkler being on.

Figure 2.1 illustrates the problem described above represented as a simple Bayesian belief Network. The various variables of the problem are represented as the nodes of the network that affect whether the pavement is slippery or not. The nodes (variables) are as follows: X_1 is the season of the year, X_2 is the probability (belief) whether it is raining, X_3 is the belief of whether the sprinkler is on and X_4 is the probability of being wet. X_5 is probability of the pavement being slippery. The network structure is determined taking into consideration the conditional independence relationships between the variables. The absence of a link between two variables represents that they are conditionally independent of each other given their parents. This is a typical example where traditional frequentist methods cannot be applied successfully

as there is very little statistical information available about the problem. The direction of the arrows in the above network defines the causal relationship amongst the variables; an arrow from X_4 to X_5 can be thought of as X_4 causes X_5. This has implications in only determining the structure of



2.1 A Simple Bayesian belief network

the network; specifically it simplifies drawing the network structure by hand. Inferences can be derived by propagating information in either order using Bayes'

theorem. There are three forms of reasoning depending on the direction in which the information is propagated. Two of them are the traditional ways of prediction and abduction: prediction occurs if in computing the probability of the pavement being slippery given it is wet, i.e., predicting the value of X_5 based on the value of X_4; abduction occurs in computing the probability of the pavement being wet given it is slippery, for example if someone slipped on the pavement, then it is probably wet, i.e., inferring the probability of X_4 given the value of X_5. Abduction propagates opposite to the direction of prediction. There is a third form, explaining away, which is especially difficult to model in rule based systems and neural networks in any natural way [PEAR00]: if it is known that the sprinkler is on, then that reduces the probability of rain.

The next step in solving the problem using BBN entails computing the joint probability distribution over the network and the marginal or conditional probability distributions over each of the variables in the network. Using equation 2.8, chain rule of probability, the following equation is obtained,

$$P(X_1,X_2,X_3,X_4,X_5) = P(X_1)P(X_2|X_1)P(X_3|X_2,X_1)P(X_4|X_1,X_2,X_3)P(X_5|X_1,X_2,X_3,X_4). \quad \text{-(2.10)}$$

As explained above, this requires an exponential amount of space for accurate representation of the probability distribution as explained above. This required space is reduced in BBNs by factoring the total joint probability in terms of the local conditional independence distributions that exist at the nodes given their parents. The conditional independence relationships are represented in the network by the absence of the links between the nodes. No link between X_1 and X_4 suggests no direct linking between the season of the year and whether the pavement is wet. The simplest and by far the most

common conditional independence relationship encoded in a Bayesian belief Network is that a node is independent of its ancestors given its parents [MITC97]. Thus here we have that X_4 is independent of X_1 given X_2 and X_3 and X_5 in independent of X_1, X_2 and X_3 given X_4. Mathematically,

$P(X_4|X_1,X_2,X_3) = P(X_4|X_2,X_3)$,

$P(X_5|X_1,X_2,X_3,X_4) = P(X_5|X_4)$,

and $P(X_3|X_1,X_2) = P(X_3|X_1)$ because of absence of link from X_2 to X_3.

Thus eq. (2.10) becomes

$$P(X_1,X_2,X_3,X_4,X_5) = P(X_1)P(X_2|X_1)P(X_3|X_1)P(X_4|X_2,X_3)P(X_5|X_4) \quad \text{--(2.11)}$$

Therefore, using conditional independence eq. (2.11) allows the joint probability to be represented more compactly. Clearly, the space requirement now grows linearly with an increase in the number of variables. The space saved in this case is minimal but in general, for n nodes, the full joint probability representation requires $O(2^n)$ and the factored representation requires $O(n2^k)$ where k is the maximum fan-in of any node in the network [HECK95].

In general, the joint probability distribution over the entire network is described as:

$$P(x_1, x_2, ..., x_n) = \prod_i P(x_i \mid pa_i), \quad \text{--(2.12)}$$

where pa_i is some set of values for the parents of Node X_i.

Therein also lies one of the major drawbacks of the BBN: the network structure is dependent on the variable order [PEAR00]. If there is an error in choosing the order,

the resulting network may not clearly reveal the conditional independence relationships and hence the savings in space and computation may be very poor.

Thus, in summary, solving any problem using BBNs boils down to 1, representing the problem as a graphical structure encoding the conditional independence relationships; 2, computing the conditional independent relationships over each of the variables and; 3, computing the joint probability distribution over the entire network. Step 1 of the above process has been illustrated here. One implication of the causal structure of the BBN is that it simplifies constructing the network by hand. The first part of the first step is to determine the variables to be modeled [HECK95]. The relationships between the variables are then drawn. Learning algorithms are used to compute the local probability distributions in steps 2 and 3. These are discussed in the next chapter.

Figure 2.1 depicts a problem model to determine whether the graphical structure formed keeping in mind the conditional independence relationships between the various elements. This is also a classic situation where no data is available about the situation before hand and all the information is actually just a set of beliefs of the occurrence of that particular state. The five variables capture the information of the problem and the decision to be made. The decision in this case to be made is whether the pavement is slippery or not.

The classic algorithms for inference to learn the probabilities in BBNs are described in the next chapter along with the MIME algorithm implemented in the thesis. A brief overview of learning in the BBNs is also provided. Learning in this case refers to learning two things from the data provided: the structure of the network and the parameters for the network. The parameters constrain the network.

Applications of Bayesian Belief Networks

BBNs are the new tool for reasoning under uncertainty. They are applied in situations where data is not available. So BBNs are used in medical diagnostic expert systems, risk assessment systems, systems to assess suitability of automobiles, and so on. They are also used for discovering causal relationships in data. The most popular applications of BBNs are the Microsoft Troubleshooter and Office Assistant. Microsoft has also developed an XML file format for BBNs [MICR]. Microsoft is also pursuing research in BBN with and applying them to solve everyday problems like filtering of junk e–mail [HECK98]. It seems likely that with the efforts of corporations like Microsoft, BBN will not be a scientific tool for solving complex problems in uncertainty but will become more practical with applications to problems affecting the general population.

CHAPTER 3
INFERENCE AND LEARNING IN BAYESIAN NETWORKS

There are two types of inference tasks in BBN. One involves the updating of probabilities of the hidden (unobserved) nodes in the network, called *belief updating* or *probabilistic inference*. The other involves computing the most likely hypothesis from a set of hypothesis given the data and is known as *belief revision* or *MAP explanation* [GUOH02]. This thesis is primarily concerned with the former--probabilistic inference. It should be noted here that algorithms for belief updating could be used for belief revision and vice versa with slight modifications. Various algorithms are used to learn the probabilities. A few are discussed in this chapter and the MIME algorithm [RANG02], which forms the basis of this work, is also presented. Inference in BBNs is different from learning: the former is computation of a probability of an hidden variable given the probabilities of the observed variables while the latter refers to determining or learning, usually from the data, the structure of the network and the parameters that constrain the probability distribution over the nodes of the network.

Inference Algorithms

A BBN is graphical representation of a joint probability distribution defined over some random variables. These variables are represented as the nodes in the network. The marginal probability of these variables (nodes) changes with the knowledge of the outcome of other variables; this probability is known as the posterior probability of the variables given the evidence. The evidence refers to the variables whose output is observed. There are two types of inference in BBNs: exact and approximate. Exact

inference refers to computing the exact probabilities of the random variables and as such is possible in a very limited number of cases: 1. all hidden nodes are discrete, or 2. all nodes, hidden and observed, are governed by a Gaussian distribution [MURP01]. Computing exact inference has been proved to be NP-hard [COOP90]. Even computing approximate inference is NP-hard [DAGU94] but the networks can still be used to compute solutions to the problems as explained below.

The process of inference in general is explained using the example from figure 2.1 [PEAR00]. There are five variables, cloudy (X_1), sprinkler (X_2), rain (X_3), wet (X_4) and slippery (X_5). The last two variables (X_4 and X_5) describe the state of an object--a pavement--and the first three variables define the conditions of the environment of that variable. The objective is to compute the value of one of the variables given the values of or more of the remaining variables. e.g., rain ($X_3 = R$) is the observed variable and the objective is to compute the probability of the pavement being wet, $P(X_5=W)$, in light of this evidence. Essentially the process involves repeated application of Bayes' theorem to compute the unknown probability.

There are various ways to solve the above problem. The most basic method would be a brute force or blind method to compute the marginal probability from the joint probability distribution,

$$P(X_5 = S) = \sum_{x_1}\sum_{x_3}\sum_{x_4}\sum_{x_5} P(X_1 = x_1, X_2 = x_2, X_3 = R, X_4 = x_4)$$

Algorithms for exact inference include an algorithm that reverses the arcs in a network to compute directly the probability of the hidden variable of interest [HOWA81] [OLMS83][SCHA88], a message-passing scheme that updates the probabilities of the hidden variables based on the probabilities of the observed variables [PEAR88], and

converting the network into a tree and utilizing the mathematical properties of the tree to compute the result to the probabilistic query [LAUR88]. One of the basic exact inference algorithms is the Sum Product algorithm and is explained below.

Sum-Product Algorithm

The algorithm was developed independently by Pearl [PEAR86][PEAR88], and Lauritzen and Spiegelhalter [SPIE86][LAUR88]; this method for probability propagation is used for inferring probability distributions in singly connected graphical models The algorithm is based on the principle that a global function can be factored into a product of simpler local functions. A particular local function may be obtained by summing over all the remaining local functions. This is illustrated using an example below [KSCH00].

The example is based on the example in Figure 2.1. Restating the problem, there are three variables that can directly affect whether the pavement is wet given by X_4: X_1 is the season of the year, X_2 is whether the sprinkler is on and X_3 is whether it is raining. If the pavement is known to be wet, then X_4 becomes the observed variable and the others are the hidden variables. Exact inference in this case would be to compute the posterior probabilities of each of the hidden variables once the outcome of the observed variable is known. Essentially, the process is just a repeated application of Bayes' theorem to compute the probabilities of each of the hidden variables. Knowing the pavement is wet means one of the two, either the sprinkler is on or it is raining, is true. However, as explained above, this algorithm defeats the purpose of the Bayesian belief networks by not using the conditional independence relationships in the network.

The probability of the sprinkler being on given the pavement is slippery is denoted by $P(X_5=1|X_4=1)$ and is computed by dividing the joint probability of the

sprinkler being on and the pavement being wet by the probability of the pavement being wet. Mathematically,

$$P(X_2 \mid X_4) = \frac{P(X_2 = 1, X_4 = 1)}{P(X_4 = 1)}.$$

The joint probability of any two variables can be computed from the joint probability over the entire distribution and summing over the variables that need to be eliminated. In this case, the joint probability of all four variables is given as $P(X_1, X_2, X_3, X_4)$. The joint probability of the pavement being wet $(X_4=1)$ and the sprinkler being on $(X_2=1)$ is computed by summing the joint probability distribution over all values of X_1 and X_3. Thus mathematically,

$$P(X_2 = 1 \mid X_4 = 1) = \frac{P(X_2 = 1, X_4 = 1)}{P(X_4 = 1)} = \sum_{x_1}\sum_{x_3} \frac{P(X_1 = x_1, X_2 = 1, X_3 = x_3, X_4 = 1)}{P(X_4 = 1)}$$

Similarly, the posterior probability of pavement being wet due to rain (X_3) is computed as

$$P(X_3 = 1 \mid X_4 = 1) = \frac{P(X_3 = 1, X_4 = 1)}{P(X_4 = 1)} = \sum_{x_1}\sum_{x_2} \frac{P(X_1 = x_1, X_2 = x_2, X_3 = 1, X_4 = 1)}{P(X_4 = 1)}$$

.The conditional independence relationships are used to overcome the computational intractability by reducing the number of variables involved in specifying each distribution.

There are two main reasons for using approximate inference even though it has also shown to be NP-hard [DAGU93]: if there is not closed form solution possible for the network or if the exact inference is impractical due to the time required. A few classes of algorithms for approximate inference are given below.

The most common class of algorithms is the *Monte Carlo algorithms*, also known as *stochastic simulation* or *stochastic sampling algorithms.* These work by generating a set of instantiations of the network, selected randomly based on the *apriori* marginal and joint probabilities of the variables. The probabilities are then approximated based on the frequencies with which the variables occur in the generated sample. The accuracy of the probabilities depends on the size of the sample and is independent of the structure of the network [GUOH02].

The first class is algorithms that generate a randomly selected set of the network based on the conditional probability at the selected nodes and then approximating the probabilities of the query variables by the frequency of their occurrence in the selected sample. The sample can also be randomly selected instantiation of the graph. These are called the *Monte Carlo algorithms*, also known as *stochastic simulation* or *stochastic sampling algorithms* [GUOH02] and are the most popular well-known approximate inference algorithms for BBN.

The second class of algorithms involves simplifying the model until an exact model becomes feasible and then executes an exact inference algorithm on the model. These algorithms are aptly known as *model simplification methods*.

The third class of approximate algorithms are based on the assumption that the majority of the probability mass exists in a relatively small fraction of the joint probability space. Based on this assumption, they compute a reasonable approximation of the probability distribution by searching for the high probability instantiation of the network. These are called *search-based methods.*

Finally there is a class of algorithms known as *loopy belief propagation*

algorithms. These algorithms have been demonstrated in computer vision and error-correcting codes. These essentially use Pearl's polytree propagation algorithm for exact inference in Bayesian networks with loops. Note that this class of BBN is different than the typical BBN, which are defined as DAGs and by definition, cannot have loops. The MIME falls under this category of algorithms. More information can be found in [MURP99].

MIME Algorithm

The mutual information minimization and entropy maximization (MIME) algorithm [RANG02] is a type of loopy belief propagation algorithm and is guaranteed to converge to a local minimum under certain assumptions. Before an explanation of the algorithm itself, a brief overview of the underlying concepts is required. There are two concepts are the basis for the MIME algorithm, these are the concepts of *Mutual Information* and *Entropy.*

Self-Information and Mutual Information

Self-Information

Self-information is the amount of information conveyed by knowledge of occurrence of an event. It is monotonically decreasing function with the probability of the event. Mathematically, it is expressed as a logarithmic function of the probability of an event. For example, let A be a random variable representing an event and let $P(A)$ represent the probability of occurrence of A, then the information conveyed by the occurrence of A, called self-information and denoted by $I(A)$ is given by

$$I(A) = -\log[P(A)] \quad \text{--(3.1)}$$

The units for measuring information depend on the base of the logarithm used. For base 2, the units are called *bits, nats* for base e (natural logarithm) and *nepers* for

base 10.

Self-information for a random variable *A* with probability *P(A)*,

1. is a monotonically decreasing of the probability of *A*.
2. can only take on non-negative values.
3. *is zero* (*I(A) = 0*) for *P(A)=1* and 1 (*I(A)=1*) for *P(A)= 0.*
4. correspondingly, for two random variables A and B,

I(A)<I(B) if P(A)>P(B)

5. and finally, if *A* and *B* are independent of each other, then *I(AB) = I(AB) =I(A)+I(B)*

Mutual Information

Mutual Information (MI) is a measure of dependence between two states of two random variables. As an explanation, let *A* and *B* be two random variables; mutual information is the information provided about the occurrence of some event $a \varepsilon A$ by the occurrence of another event $b \in B$. Alternatively, mutual information is the amount of reduction in uncertainty about the occurrence of one event ($a \in A$) by the occurrence of another event ($b \in B$). Mathematically this is

$$MI(A,B) = \sum_{a \in A, b \in B} P(A=a, B=b) \log \frac{P(A=a \mid B=b)}{P(A=a)} \quad \text{--(3.2)}$$

The minimum value of the expected MI is 0 and occurs when the second event (*b* here) provides no information about the first event (*a* here); this condition occurs only when the two events are mutually independent.

Entropy

Entropy is a measure of randomness of a variable--the greater the amount of

randomness in a variable, the greater its entropy. Formally,

Let X be a random variable and let $x \in X$ be an event; andlet $P(X)$ be the probability distribution over X. The entropy of X, denoted by H(X) is defined as

$$H(X) = -\sum_{x \in X} P(X = x) \log P(X = x) \quad \text{--(3.4)}$$

or

$$H(X) = E_x(-\log P(X)). \quad \text{--(3.5)}$$

Thus, the entropy of a random variable X is the expected value or the average self-information of the information of the variable.

Since entropy is the measure of randomness of a variable, it is logical that entropy for the variable will be maximum when randomness of the variable is maximum. If a variable can have M different values, it will have maximum randomness when each state is equally likely.

Conceptual Explanation of the MIME Algorithm

MIME is an algorithm for Bayesian belief Propagation (BBP) that is guaranteed to converge [RANG02] on graphs with loops. The algorithm is a combination of the results of two previous works: 1. a connection between the conventional BBP algorithms and some methods based in statistical physics, specifically the Bethe and Kikuchi free energy functions [YEDI01]; and 2. double loop algorithms, developed by Yuille [YUIL01], guaranteed to minimize the above mentioned energy functions while maintaining the semblance to the original BBP algorithms. The MIME works, as briefly mentioned above, by minimizing the mutual information and maximizing the entropy to derive an energy function that is equivalent to the Bethe free energy function [RANG02]. The statement of the MIME principle is minimize the mutual information between pairs

(pairwise mutual information) of nodes and maximize the entropy for each variable (marginal entropy) using the marginal and link function expectations and simultaneously satisfying the joint probability constraints. The minimization of mutual information tries to make each pair of nodes as independent as possible, and the maximization of entropy puts each of the random variables in the most uncertain state possible. This process makes the network as unbiased as possible. However, when constrained by link function expectations derived from the problem data and the joint probability satisfaction requirements, it leads to the most likely state for the variable given the data for the problem.

MIME: Mathematical Explanation

The MIME cost function is a function of the joint probability, $p_{ij}(x_i,x_j)$[11], the marginal probability, $p_i(x_i)$[22], and the Lagrange parameters (γ_{ij} and λ_{ij}). The mathematical equation of the MIME is as follows.

$$F_{MIME}(\{p_{ij}, p_i, \gamma_{ij}, \lambda_{ij}\}) = \sum_{ij:i>j} \sum_{x_i x_j} p_{ij}(x_i, x_j) \log \frac{p_{ij}(x_i, x_j)}{p_i(x_i) p_j(x_j)} + \sum_i \sum_{x_i} p_i(x_i) \log p_i(x_i)$$

$$- \sum_{ij:i>j} \sum_{x_i x_j} p_{ij}(x_i, x_j) \log \psi_{ij}(x_i, x_j) - \sum_i \sum_{x_i} p_i(x_i) \log \psi_i(x_i) - \mathrm{K} \sum_i \sum_{x_i} p_i(x_i) \log p_i(x_i)$$

$$+ \sum_{ij:i>j} \sum_{x_i x_j} \lambda_{ij}(x_j) \left[\sum_{x_i} p_{ij}(x_i, x_j) - p_i(x_i) \right] + \sum_{ij:i>j} \sum_{x_i x_j} \lambda_{ji}(x_i) \left[\sum_{x_j} p_{ij}(x_i, x_j) - p_i(x_i) \right]$$

[1] $P_{ij}(x_i,x_j)$ is the joint probability of the nodes i and j being in states x_i and x_j. x_i and x_j are the states that the variables can be in. In this case, these are 0 (node not part of clique) and 1 (node part of clique).

[2] $P_i(x_i)$ is the marginal probability of node i being in sate x_i.

$$+\sum_{ij:i>j}\gamma_{ij}\left(\sum_{x_i x_j} p_{ij}(x_i,x_j)-1\right) \quad \text{--(3.5)}$$

The above is the MIME cost function and is equivalent to the Bethe free energy function when the following constraints of the relating the joint probabilities with the marginal probabilities are met [RANG02]:

$$\sum_{x_i} p_{ij}(x_i,x_j) = p_j(x_j) \quad \text{--(3.6a)}$$

$$\sum_{x_j} p_{ij}(x_i,x_j) = p_i(x_i) \quad \text{--(3.6b)}$$

$$\text{and } \sum_{x_i x_j} p_{ij}(x_i,x_j) = 1 \quad \text{--(3.6c)}$$

Note these constraints are the same as for joint probability. Marginal probability for a variable can be computed by marginalizing over the joint, i.e., summing over all the other variables in the joint distribution as explained in the previous chapter. The last three term of the energy function are to satisfy these constraints.

Mutual information in this notation is defined as

$$MI_{ij} = \sum_{x_i,x_j} p_{ij}(x_i,x_j)\log\frac{p_{ij}(x_i,x_j)}{p_i(x_i)p_j(x_j)} \quad \text{--(3.7)}$$

Clearly, the mutual information will be minimized when the constraints in eq. (3.6) above are met. These constraints can be satisfied by the variables being independent of each other. Minimizing the pairwise mutual information and maximizing the marginal entropy tries to put the system in a state of "equilibrium" (an unbiased state); however, the pairwise and singleton link functions do not let this happen. The

pairwise and singleton link functions are the third and the fourth terms in eq. (3.5) above, i.e.,

$$\sum_{ij:i>j} \sum_{x_i x_j} p_{ij}(x_i, x_j) \log \psi_{ij}(x_i, x_j) \quad \text{--(3.8a) (pairwise)}$$

$$\sum_{i} \sum_{x_i} p_i(x_i) \log \psi_i(x_i) \quad \text{--(3.8b) (singleton)}$$

The second term in the function represents the marginal entropy.

$$\sum_{i} \sum_{x_i} p_i(x_i) \log p_i(x_i) \quad \text{--(3.9)}$$

This is the function that needs to be maximized in the cost function. The above term is annihilated by the use of the following term,

$$-\kappa \sum_{i} \sum_{x_i} p_i(x_i) \log p_i(x_i) \quad \text{--(3.10)}$$

If the value of the parameter $\kappa = 1$ in eq. (3.10), then it becomes negative of eq. (3.9). Clearly, in such a situation, the entropy term is eliminated or annihilated. For this reason the parameter κ is known as the annihilation parameter. The annihilation of the entropy term causes an increase in the rate of minimization of mutual information.

The Lagrange parameters, λ and γ, enforce the above constraints between the marginal and the joint probabilities in the Bethe free energy and the MIME function. These disappear once the constraints are met [RANG02]. These pairwise and singleton link functions are derived from the problem being solved and so they depend on the problem the network is slated to solve. The formulation for clique is explained in chapter 5.

The MIME Algorithm

As explained above, the key to minimizing the energy function is to satisfy the constraints, which are actually the relationship between the marginal and joint probabilities of the random variables in the model. Rangarajan and Yuille [RANG02] derived a family of algorithms for belief propagation; the basis of all the algorithms is the same.

The most important part of the algorithm is the terms for updating the marginal and joint probabilities. These are,

$$p^{new}{}_{ij}(x_i, x_j) = p_{ij}{}^{old}(x_i, x_j) \sqrt{\frac{p_i{}^{old}(x_i)}{\sum_{x_j} p_{ij}{}^{old}(x_i, x_j)}}, \text{ and} \tag{3.11a}$$

$$p_i{}^{new}(x_i) = \sqrt{p_i{}^{old}(x_i) \sum_{x_j} p_{ij}{}^{old}(x_i, x_j)} \tag{3.11b}$$

Eq. (3.11a) is the joint probability update equation defined in terms of the marginal probability and eq. (3.11b) is the marginal probability update equation defined in terms of the joint probability. Summing eq. (3.11a) over x_i yields eq. (3.11b) showing that the two equations, when updated simultaneously, satisfy the constraints of the energy function mention in eq. (3.6).

A very simplified version of the algorithm is described below. This is the version that was implemented in the BBN and used to run all the benchmarks. The algorithm for the update consists of two loops for constraint satisfaction preceded by initializing of all

the variables. The algorithm is shown with the free parameters--d and x--set to 0; this leads to the parameters settings of $q_i=n_i$ and $r_i=1$, where n_i is the number of neighbors of node *i*.

N is the number of variables (nodes in the network) and *M* is the number of states for each node. For the clique problem *N* is the number of nodes in the graph and M = 2 (0 or 1--either the node is part of the clique or not) [RANG02].

1. Initialize the joint and marginal probabilities $\{p_{ij}, p_i\}$. These can be defined according to any prior information about the problem domain, for this thesis however, since there was no information available, these were set to be equally likely and defined as (1/M) and (1/MN).

2. Outer Loop: The outer loop consists of initializing the variables for constraint satisfaction. The computation of the marginal probabilities by summing over the joint probability is done here. The joint probability is updated as

   a. $$p_{ij}(x_i, x_j) \leftarrow p_{ij}(x_i, x_j)\psi_{ij}(x_i, x_j)$$

where $\psi_{ij}(x_i, x_j)$ is the pairwise link function.

   b. $$p_i^{new}(x_i) \leftarrow [(p_i^{old})^{n_i}(x_i)]\psi_i(x_i)$$

where $\psi_i(x_i)$ is the singleton link function.

3. Inner Loop: This is the where all the constraint satisfaction is done using the simultaneous update equations from equation (3.12). This is done in two steps.

   a. Update $p_{ij}(x_i, x_j)$ and $p_i(x_i)$ as follows:

$$p_{ij}^{new}(x_i, x_j) \leftarrow p_{ij}^{old}(x_i, x_j)\sqrt{\frac{p_i^{old}(x_i)}{\sum_{x_j} p_{ij}^{old}(x_i, x_j)}} \text{ and}$$

$$p_i^{new}(x_i) \leftarrow \sqrt{p_i^{old}(x_i)\sum_{x_j} p_{ij}^{old}(x_i, x_j)}$$

   b. Update $p_{ij}(x_i, x_j)$ and $p_j(x_j)$ as follows:

$$p_{ij}^{new}(x_i, x_j) \leftarrow p_{ij}^{old}(x_i, x_j)\sqrt{\frac{p_j^{old}(x_j)}{\sum_{x_i} p_{ij}^{old}(x_i, x_j)}}, \text{ and}$$

$$p_j^{new}(x_j) \leftarrow \sqrt{p_j^{old}(x_j)\sum_{x_i} p_{ij}^{old}(x_i, x_j)}$$

   c. Normalize $p_{ij}(x_i, x_j)$: this satisfies the third constraint eq. (3.7c).

$$p_{ij}^{new}(x_i, x_j) \leftarrow \frac{p_{ij}^{old}(x_i, x_j)}{\sum_{x_i, x_j} p_{ij}^{old}(x_i, x_j)}$$

   End Inner Loop

End Outer Loop

3.1 MIME algorithm simplified.

Complexity Analysis of the Algorithm

The MIME is a double loop algorithm, however both the loops are bounded to a maximum limit, this reduces its complexity under $O(n^2)$; the details are explained below.

The outer loop is responsible for minimizing the MI and maximizing the ME. Computation of the MI is based on the marginal and joint probabilities and ME is based solely on the marginal probabilities of the nodes. Obviously, computing the joint probability has a higher complexity than the marginal probability. In the MIME, the joint probability is computed over the nodes of the graph that are connected to each other and so there are $|E|$ joint probabilities where $|E|$ is the number of edges in the graph; thus the joint probability updates are of the order $O(|E|)$. The marginal probabilities are computed for each of the nodes in the network and so become $O(N)$ where n is the number of nodes in the graph.

The inner loop consists of updating the joint and the marginal probabilities. Thus for each iteration of the inner loop, the complexity becomes $O(|N|+|E|)$, which is $O(|E|)$ when $|E|>|N|$. The total complexity of the inner loop is the complexity just derived multiplied with the number of iterations of the inner loop (say B)[3], i.e. $O(B(|N|+|E|))$. There is an upper bound on the number of iterations of the inner loop, so B is a constant.

Similarly, in the outer loop, since both the updates are done in the outer loop, then the complexity per iteration of just the outer loop is complexity of the outer loop functions multiplied with the complexity of the inner loop, $O(\{|N|+|E|\}B\{|N|+|E|\})$. For the complexity of the entire loop, the complexity obtained is multiplied with the number of outer loop iterations (A), which is a constant

since there is an upper bound on the number of outer loop iterations also. This can be simplified further when $|E| > |N|$, and so $O(|N| + |E|) = O(|E|)$. Therefore the complexity of the algorithm is

$$O(A(E + B(E)))$$

Since A and B are constants, this further reduces to:

$$O(|E| + |E|) = O(|E|)$$

which is the complexity of the algorithm in the case when $|E|>|N|$.

[3] The number of inner loop and outer loop iterations is set arbitrarily and serve as an upper bound to satisfying the constraints.

CHAPTER 4
MAXIMUM CLIQUE AND OTHER NP COMPLETE PROBLEMS

This chapter provides an overview of NP-completeness, NP-complete problems in general and the Maximal Clique Problem in particular. It also explains a few common approaches to solving the Maximum Clique Problem, including the approaches followed in the benchmark algorithms from the second DIMACS challenge on NP-complete problems.

Polynomial Time Problems

Computational problems may be divided into 4 classes [HORO98]. These classes are called the *P, NP, NP-complete* and *NP-hard*. The simplest relationship amongst these classes is between the *P* and *NP* classes. *P* or Polynomial time problems are those that can be solved deterministically in polynomial time i.e. $O(n^k)$ for some constant *k,* where *n* is the input length [HORO98]. *NP* or *Non-deterministic* time problems are the ones that can be solved non-deterministically in polynomial time. All problems that are *P* are also *NP*, that is, any problem that can be solved in polynomial time can definitely be solved deterministically in polynomial time [CORM01]. Thus it is known that,

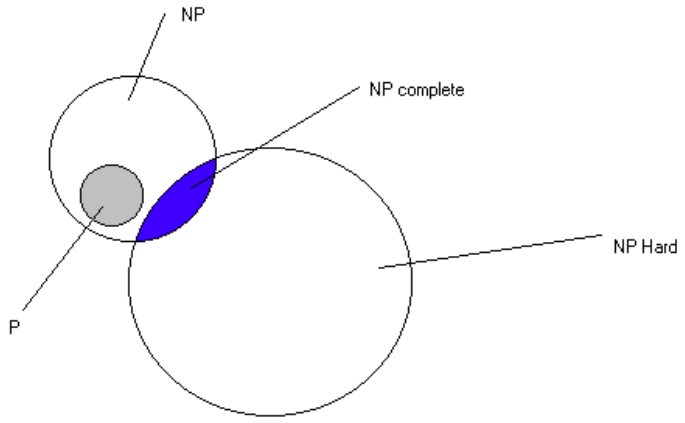
$$P \subseteq NP \qquad (4.1)$$

However, perhaps the most famous unsolved problem in computer science is whether every *NP* problem is also a polynomial i.e. if *P* is a proper subset of *NP* [CORM01] or in other words is *P=NP*? (or equivalently is *P≠NP?)*

Before explaining the concepts of NP-hard and NP-complete, it is important to list the definition of satisfiability and CNF satisfiability. Satisfiability of a Boolean formula is to determine if the formula is true for some assignment of truth-values to the variables in the formula. CNF (Conjunctive Normal Form) satisfiability is defined as the satisfiability problem for CNF formulas.

Reducibility is defined as follows. Let *L1* and *L2* be problems, then *L1* is said to be reducible to *L2* iff *L1* can be solved deterministically in polynomial time by the same algorithm that solves *L2* deterministically in polynomial time [HORO98].

A problem is said to be NP-hard iff satisfiability reduces to the same problem, meaning that. Formally, a problem *L* is said to be NP-hard iff satisfiability reduces to *L*. NP-complete problems are problems that are both NP-hard and NP. This relationship is depicted in figure 4.1.



4.1 The four classes of problems.

Some Important NP Complete Problems

NP-complete problems fall in two categories, decision and optimizations. Some well-known NP-complete problems are the Traveling Salesman Problem, Graph Coloring, problem and Hamiltonian cycle problem. The maximum clique is also an NP-complete problem.

The Maximum Clique Problem

The maximum clique problem is defined as follows: Let $G=(V,E)$ be an undirected graph where $V=\{1,2,\ldots,n\}$ is the set of the vertices and $E \subseteq V \times V$ is the set of edges of the graph, then a clique of the graph is defined to be a sub graph $S \subseteq G$ such that all the vertices of S are pairwise adjacent; in other words, every vertex in the subgraph is connected with every other node in the subgraph. The maximum clique problem is to find a sub graph S as defined above of the highest possible cardinality. Cardinality refers to the number of elements in a set and in this case is denoted by $|S|$ [BOMZ99]. Mathematically,

$\omega(G) = \max\{|S| : S \text{ is clique in the graph}\}$

where $\omega(G)$ is known as the *clique number* of the graph G and gives the size of the maximum clique.

The maximum clique problem has other equivalent formulations. One such formulation is the *maximum independent set* problem. An *independent set* of a graph is defined as a subset of V, all elements of which are pairwise non adjacent. The independent set problem is also known as the *stable set* or *vertex-packing* problem. A complement graph of $G=(V,E)$ is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(i,j) | i,j \in V, i \neq j$ and

$(i,j) \notin E\}$. Clearly the maximum independent set problem on the complement graph is the same as the maximum clique problem on the original graph.

It is important to differentiate between the maximum clique and the maximal clique problems. Both the maximal clique and the maximum clique are independent sets of the graph. A maximal clique is a clique that is not a subset of any other clique of the graph; the maximum clique is a maximal clique that has the highest cardinality or clique number. Thus, a maximal clique is a clique that is not necessarily the largest clique in the graph and this is the problem that is the topic of this thesis.

Complexity

The maximum clique problem is proved to be NP-complete [KARP72]. By definition, this problem, cannot be solved optimally for all graphs unless *P=NP*. Therefore, most research has been done on creating approximation algorithms for the problem. Even approximating the maximum clique in polynomial time has been proved to be impossible so far [CRES91]. The best results for approximation of the maximum clique problem are that it can be approximated in $O(|V|/(\log|V|)^2)$ [BOPP92]. Finally, it has been shown that the maximum clique size cannot be approximated by a polynomial time algorithm within a factor of n^{ε} ($\varepsilon>0$), unless *P=NP* [BOMZ99]. The best possible approximation to the maximum clique problem that can be achieved has been shown to be an approximation ratio of $n^{1-o(1)}$. Taken together this means that the maximum clique problem is a very hard problem to solve.

Problem Formulations

A very crucial aspect of NP-complete problems is choosing the right formulation. The maximum clique has various equivalent formulations in terms of an integer

programming or a continuous nonconvex optimization problem [BOMZ99]. A few of the common integer formulations are listed here.

Integer Formulations

Edge Formulation

The edge formulation is the simplest formulation of the problem and is defined simply as the maximum clique of a graph *G=(V,E)* is given by:

$$\max \sum_{i=1}^{n} w_i x_i \qquad \text{--(4.1)}$$

$$\text{s.t. } x_i + x_j \le 1, \forall (i,j) \in \bar{E}$$

$$x_i \in \{0,1\}, i = 1,...,n$$

where W is the weight vector[1] and $w_i \in W$.

Independent Set Formulation

An equivalent formulation of the above is in terms of the maximum independent set problem. The maximal clique of a graph *G=(V,E)* is given by:

$$\max \sum_{i=1}^{n} w_i x_i$$

--(4.2)

s.t. $\sum x_i \le 1, \forall S \in \Sigma$: where Σ is the set of all maximal independent sets of G

$$x_i \in \{0,1\}, i = 1,...,n$$

The edge formulation of eq. (4.1) can be rewritten as a minimization problem in terms of a quadratic zero-one problem.

$$\min f(x) = -\sum_{i=1}^{n} x_i \qquad \text{--(4.3)}$$

$$\text{s.t. } x_i + x_j \le 1, \forall (i,j) \in \overline{E}, x \in \{0,1\}^n$$

The formulation of the problem on which the MIME link functions are based is an equivalent formulation of the maximum clique as a global quadratic zero-one problem and is defined as:

$$\begin{aligned} &\min f(x) = x^T Ax \\ &s.t. x \in \{0,1\}^n \end{aligned} \qquad \text{--(4.4)}$$

where $A = A_{\overline{G}} - I$ *and I is the identity matrix.*

In the above formulation, eq. (4.4), the off-diagonal elements of the matrix *A* are the same as the off-diagonal elements of the adjacency matrix of the complement graph denoted by $\overline{G}$. The computations for the maximum clique are performed on this matrix; this matrix becomes increasingly sparse as the density of the original graph increases. Therefore, this formulation is suited for very high-density graphs. Since the MIME link functions are based on this formulation, the MIME based BBN is ideally suited for computation of the maximum cliques for dense graphs.

Algorithms for the Maximum Clique Problem

There are two basic approaches in algorithms for the maximum clique problem; exact algorithms and heuristics. Exact algorithms are not very practical for this problem, as it has been proved to be NP-complete. The results deteriorate very fast for larger graphs. The main area of interest is the heuristics used to solve the same problem: Moreover the BBN is also a heuristic approach to the same problem. The section provides a broad overview of exact algorithms and enumerates a few heuristics in detail.

Exact Algorithms

One class of exact algorithms is known as enumerative algorithms; these algorithms list all the cliques in a graph and are the subject of this section. The first

algorithm of this type was based on an inductive method that consisted of identifying all the cliques of a special graph with no more than three cliques and then reducing the problem in general graphs to this special case [HARA57]. Some other well known exact algorithms use the *vertex sequence* or *point removal method* [AUGU70] and the *backtracking method* [AKKO73].

A simple and effective exact algorithm for computing the maximum clique of a graph was proposed Pardalos and Carraghan [CARR90]. The algorithm was used as a benchmark in the second DIMACS Challenge [JOHN96]. The algorithm finds the maximum clique containing an arbitrary vertex v_1. This vertex is then excluded from any further considerations of cliques since it is not possible to find a clique larger than the one already found containing v_1. The algorithm then takes the next vertex and repeats the process. The algorithm forms the basis of an exact parallel algorithm [PARD99] that uses the greedy randomized adaptive search procedure (GRASP) for maximum independent set [FEOT95]. The parallel algorithm is used to obtain good starting solutions. Another algorithm is can be found in [OSTE02].

Heuristics

The computational complexity of the max clique, along with it being hard even to approximate has been the cause of a lot of research in devising efficient heuristic solutions to the same. Heuristics do not offer any guarantee of performance but are still of interest in practical applications due to various reasons. The reasons are similar to devising and using approximate inference algorithms for BBNs even though it is known to be NP-hard.

Heuristics solve the same problem in different ways and for different ways. Therefore, the only way to evaluate their performance is through experimentation. Still it

may not be possible to compare the results obtained from one heuristic with the results from another heuristic. Due to this reason, a set of benchmark graphs and algorithms was proposed in 1993 in conjunction with the second DIMACS challenge. A few of the common heuristics are listed below.

Sequential Greedy Algorithms. These are based on the greedy programming approach. There are typically two approaches for this: Either repeated addition of a vertex to a set of nodes comprising a partial clique *(Best in)* or the repeated deletion of a node from a set of nodes that are not the clique *(Worst out)* [BOMZ99]. Adding or deleting a node to or from the set of nodes is based on the suitability of the node for being in the clique. One criterion for best in may be a node that has the maximum number of neighbors (degree) and for likewise for worst out, the node with the smallest degree. The sequential greedy method essentially finds a maximal clique (which is a local minima) and stops searching. It is quite likely that the global minima is very close to the local minima obtained from the greedy sequential search and can be found by further searching. This is what the next heuristic essentially does.

Local Search Heuristics. These heuristics are based on the approach defined above. Essentially, the search heuristic depends on the structure of the network and the sequential heuristic used to find the local minima. A local search heuristic may be as simple as including some random factors to generate a clique and then searching repeatedly with different random factors.

Simulated Annealing. This particular heuristic is a type of advanced search heuristic. It is a common heuristic used in many different applications including training neural networks. The heuristic aims to minimize a free energy function for the system.

Simulated annealing is a randomized search algorithm based on the physical process of annealing. It is applicable to systems where an energy function can be used to describe the system [KIRK83]

Artificial Neural Networks. Artificial Neural networks (ANN) in more than one sense can be considered the predecessor of BBN. These are massively parallel, distributed systems that have been applied to many NP Complete and uncertain reasoning problems. Some of the important areas of applications of ANNs are pattern recognition, learning and adaptation, data mining [MITC97] and universal approximation.

One class of ANNs applied to the maximal clique problem is known as Hopfield networks [JAGO95]. These networks have been applied to other complex combinatorial optimization problems like the Traveling Salesman Problem also.

Benchmark Approaches

The MIME BBN was compared against the benchmark programs part of the DIMACS challenge on the DIMACS benchmark graphs. There are two programs that are provided part of the benchmark suite: the dfmax.c and the dfclique.c. The first one is the exact optimal algorithm based on branch and bound and the second is a semi-exhaustive greedy approach.[JOHN96]

Applications of the Maximum Clique Problem

The maximum clique is a combinatorial optimization problem and has many practical applications. Many real world can be broken down completely or in part to the maximum clique problem. One of the application areas is coding theory where the problem of finding the largest possible binary code to correct errors in binary words of a given size can be reduced to finding the maximum clique on certain graphs. [BOMZ99]. One major area where solutions to the maximum clique problem are directly applicable is

computer vision and pattern recognition. Future work for the thesis is to apply the maximum clique to computer vision.

CHAPTER 5
IMPLEMENTATION DETAILS

This chapter explains the design and implementation aspects of the thesis. Adapting the MIME cost function for the Maximum Clique Problem is explained first followed by the design details and finally the implementation details.

Adapting the MIME

The MIME involves minimizing the mutual information and maximizing the marginal entropy constrained by the singleton and pairwise link expectation functions. The link expectation functions are specified according to the data constrains of the problem.

The MIME pairwise and singleton link expectation functions in the MIME were mentioned explicitly in equation (3.10) and are reproduced here:

$$\sum_{ij:i>j} \sum_{x_i x_j} p_{ij}(x_i, x_j) \log \psi_{ij}(x_i, x_j) \qquad \text{--(5.1a) (pairwise)}$$

$$\sum_{i} \sum_{x_i} p_i(x_i) \log \psi_i(x_i) \qquad \text{--(5.1b) (singleton)}$$

The adaptation of these functions for the MIME cost function for maximum clique is based formulation for computing the maximum clique given by Bomze *et al* [BOMZ99] explained in the previous chapter eq. (4.4): This problem formulation gives the following clique cost function:

$$E(V) = \min\left(\sum_{ij} G_{ij} v_i v_j - \sum_i {v_i}^2 \right) \qquad \text{--(5.2)}$$

The above cost function is represented for the BBN as follows:

$$E(\{p_{ij}, p_i)\} = 2\sum_{ijab} p_{ijab} a.b.G_{ij} - \sum_{ia} a.p_{ia} \quad \text{--(5.3)}$$

$$a, b \in \{0,1\}, i > j$$

The above cost function is the clique cost function for the MIME BBN.

a and *b* refer to the states that the network can be in. If a node is in state 0, it means the node is not part of the clique, and if it is state 1, it means that the node is included in the clique.

p_{ijab} is the joint probability of the node *i* being in state *a* and node *j* being in state *b* at the same time. *i* and *j* refer to the nodes in the complement graph. Clearly, two nodes together can be four different states:

1. *a=0 and b=0*--neither of the two nodes are in the clique.
2. *a=0 and b = 1*--node b is in the clique and *a* is not.
3. *a=1 and b=0*--node *a* is in the clique and *b* is not.
4. *a=1 and b=1*--both nodes *a* and *b* are in the clique.

Of the above four states, state 4 must always be 0. This is so because two nodes that are connected in the complement graph do not have an edge connecting them in the original graph and hence at most one of the two can be in a clique at any given time. This constraint could have been hardcoded by explicitly assigning the value 0 to all $p_{ij11}=0$. However, this is not done and there have been no results to show this to be a cause for concern.

The above result for the minimum value for the first term of the cost function reduces the cost function for the clique to the minimization of the second term of eq. (5.3), $- - \sum_{ia} a.p_{ia}$ or maximization of $\sum_{ia} a.p_{ia}$; this terms results in the clique when all other constraints are met.

The pairwise and link expectation functions for the MIME for clique are derived from eq. (5.3) above. Essentially, a formulation for the link functions is required that reduces them to the terms in eq. (5.3).

The link functions are listed in eq. (5.1 a) and eq. (5.1 b); assigning the following values to the ψ terms reduces the terms to the terms of the cost functions:

$$\psi_{ij}(x_i, x_j) = e^{-abG_{ij}} \quad \text{--(5.4)}$$

$$\psi_i(x_i) = e^{-a} \quad \text{--(5.5)}$$

Eq. (5.5) reduces to the second term in the clique cost function and gives the size of the clique.

Design and Implementation

The following section describes the approach followed in design and implementation of the code. A very general overview of the design is provided first followed by a detailed explanation of the design and the implementation. The design section deals with the design of a general BBN and not one specifically for the maximal clique problem. The design considerations specific to solving the maximal clique are explained in the implementation section because it would not make sense to explain them without listing the reasons for making them first.

Design

The main design issue of the thesis was to implement a general purpose, object oriented BBN using the MIME for belief propagation that could be easily adapted for solving other problems. The reason for making it object oriented was to make it easier to add on to the code and also so as to provide a clean interface for extending the program. A Graphical User Interface (GUI) for the BBN was developed to enhance the user interface. As far as possible, the JAVA naming convention [BEUS] was followed in the design and the implementation, i.e. the constants were all uppercase, the names of classes were capitalized and, variable names started with a lower case letter with each subsequent word capitalized.

The BBN is a graphical model and very cleanly and logically breaks down into a group of nodes connected by edges like any graph. Based on this, there are two main classes: Node and Network. The class Node encapsulates the data items that are part of the each individual node (vertex) in the network. This data includes marginal probabilities, the singleton link data and functions of the nodes in the network. The class Network encapsulates the nodes, which are instances of the class Node and the functions pertaining to the nodes, the edges and network as a whole. Thus it includes the functions setting the structure or topology of the network, initializing the network in terms of setting the parameters and for belief Propagation or inference in the network. The class also contains a function to extract the clique from the data provided by the BBN. Among the essential attributes of the class are the parameters of the BBN, the joint probability and, the pairwise link function.

A listing of the functions and variables in both classes in UML format is given below in figure 5.1 and 5.2; most of the data elements and the functions are self-explanatory while some of the more important ones are explained in detail.

One important difference between the design and implementation of the code from the mathematical formulation of the MIME algorithm is in the notation followed in the two. In the design and implementation, there are essentially two types of indices: 1. going over the states of the nodes denoted by a and b and 2.going over the nodes themselves, denoted by i, j and, k. Thus, p_{ia} refers to the marginal probability of the node i being in state a. Similarly, for joint probability, p_{ijab} refers to the joint probability of node i being in state a, and node j being in state b. In the original notation, these two terms were represented as $p_i(x_i)$ and $p_{ij}(x_i,x_j)$ respectively.

An explanation of the attributes and functions of the class Node are provided below followed by an explanation for the class Network.

-ID is to give the node a unique id, each node has its own id number, which is a number generated serially and gives the order in which the node was created.

-NUM_STATES is a constant and gives the number of states the node can be in. Thus the NUM_STATES changes for according to the problem being solved. This variable is a constant and does not change the value once it has been set. In the case of the Clique, it indicates whether the node is part of the clique (value 1) or not (value 0) and so its value is hard coded to 2.

prob[] and rho[] are arrays that hold the values of the probabilities of the node being in the clique or not. rho[] holds the values of the probability before updating (pi^{old})

according to the MIME algorithm. prob[] holds the new probability, i.e. the probability after the updating (p_i^{new}).

psi[] holds the values for the singleton functions

updateProbOuter() is the most important function of the class. The function has two forms, one if for a special case, used to speed up the computation and performs two tasks, checks for and corrects the underflow error and updates the marginal probability

Class Node
-nodeID: int {static} -ID:int {CONSTANT} -NUM_STATES:int {CONSTANT} -EXPMIN:double -prob[]:double -rho[]:double -psi[]:double -q:double -r:double -xi: double -numNeighbors:int
<<constructor>> +Node() <<initialization functions>> +initializeRho():void +initializePsiAndProb():void <<accessors>> +getPsi(int):double +getXi():double <<mutators>> +setQ(double, double); +setR():void +setNumNeighbors(int):void +setProb(int,double):void +setRhoToProb():void <<utility functions>> +sumProb():double +updateProbOuter(messProd:double[][]):void +updateProbOuter(beta:double, messProd:double[][]):void +getSumOverAllProb():double +divideProb(sumPi:double):void +divideProb(a:int, sumProb:double):void <<print functions>> +printProb():void

5.1 UML of class Node

within the outer loop according to the algorithm described in chapter 3 i.e. according to the following equation.

$$p_i(x_i) = \rho_i(x_i)^q \psi_i(x_i)^{\beta r} e^{q-1} \qquad \text{--(4.1)}$$

The messProd is used to pass messages between the edges and speed up the algorithm. Details on message passing can be found in the MIME paper [RANG02].

The class Network is naturally more complex than the Node as it deals with the entire network, the free parameters, setting the structure of the network, belief propagation and so on. The network computations can be classified into three stages: 1. the initialization stage 2. the constraint satisfaction or belief propagation stage and 3. the post constraint satisfaction stage. The initialization stage basically sets the network up; initialize the variables, define the structure, allocate memory and so on. The BP stage performs the computations and satisfies the marginal and joint probability constraints in the network. The final stage is the one that uses the information given by the network and essentially interprets it to solve the problem. This stage is more like the data processing stage and does not have anything to do with the BBN *per se*; the BBN gives probabilities in the form of raw numbers--proper interpretation is required in order to solve the problem. The required interpretation is done in the above-mentioned phase.

The functions in the class Network were defined to follow these categories. The

are two main classes of functions in this class and are grouped accordingly: the initialization functions and the belief propagation functions.

The initialization phase for the network consists of initializing the variables of the BBN that includes the joint probabilities, the parameters of the network. This phase also sets the structure (topology) of the network, which is actually the topology of the graph, and sorts the nodes according to the number of neighbors that they have. This is a kind of bias on the network as the nodes that have more neighbors have a higher probability of BP and the post constraint satisfaction stages are combined into one stage. Thus, there

class Network
-isBeta1:Boolean -temp:int -NUM_NODES:int -NUM_STATES:int -EPS:int -nodes: Node[] -G:int[][] -Gorig:int[][] -A:double[] -order:int[] -neighbors:int[] -totalSize:int -seed:int -pij:double[][][][] -psiij:double[][][][] -sigma:double[][][] -sumPijb:double -sumPija:double -sumPijab:double -annihilate:double -delta:double -beta:double -constraintError: double -innerThreshold: double -outerThreshold: double -totalInner:int -totalOuter:int -totalLoop:int -sumPi: double -eps:double -randu:double -lengths:int -lengthT:int[] -lengthsPointer:int -rowIndex:int[][], -columnIndex:int[][] -piaInt:double -nearInteger:int -Link: double -cliqueSize:int -clique:int[] -cliqueCount:int -array:int[] -neighOrder:int[] -messages: double[][][] -messProd: double[][] -totalIterations:int-totalSkips:int -cij:double[][]

```
      -loopIterationsVector: Vector<int>
      -skipsVector: Vector<int>
      -EDGE_PROB: double
      -TMARGINAL: double
      -TJOINT: double
```

```
      <<private utility functions>>
      -initializeLink():void
      -getLink(int, int):double
      -rowColumnIndex(rowIndex:int[][], columnIndex:int[][],
lengthT:int[], lengthsPointer:int&):void
      -setTopology():void
      -setTopologyClique():void
      -setAllQ():void
      -setAllR():void
      -rowColumnIndex():void
      -getNeighbors():void
      -getTotalSize():int
      -initializePsiAndPij():void
      -loops():void

      <<CLIQUE METHODS>>
      -sortNeighbors():void
      -quickSort(array:double[], N:int, order:int[]): void
      -updateMessProdOuter():void

      <<public functions>>
      <<constructor>>
      +Network();
      +Network(N:int,ANNIHILATE:double, EDGE_PROB:double, BETA:doub
DELTA:double, OUTER:int, INNER:int, CONSTRAINT_THRESHOLD:double,
INTEGER_THRESHOLD:double, SEED:int);
      +initialize():void
      +start():void

      <<print functions>>
      +printState():void

      +cliqueExtract(order:int[], N:int): int
      +printEnd():void
      +printArray(array:double[], N:int): void
      +printArray(array:int[],N:int): void
      +printProb():void
```

5.2 UML of class Network

giving a larger clique than the nodes with fewer neighbors.

Belief propagation is performed in a function called loops(); the function implements the double loops for belief propagation at the heart of the MIME algorithm.

Belief propagation is achieved according to the MIME principle by satisfying the constraints between the marginal probability and the joint probability inside the two loops defined in the algorithm in chapter 3. The outer loop repeats until either 1. the marginal probability becomes greater than a pre defined threshold 2. the required number of outer loop iterations are completed. The default threshold for the marginal probability was set at 0.01. This constraint on the marginal probability is satisfied only when all the marginal probabilities are either less than the threshold or greater than the complement of the threshold. This essentially means the marginal probabilities reach very close (within 0.01) of an integer solution.

The inner loop keeps repeating until either the constraint error falls below the threshold constraint error or the predefined number of inner loop iterations have been completed.

The constraint error is a measure of the degree of independence of the variables in the network. It is computed as the sum of the squares of the difference between the marginal probabilities computed by marginalizing over one variable from the joint probability and the actual marginal probabilities for each of the variables. The threshold error is defined by the user and the default value was set at 1.0e-14.

One of the most straightforward ways of achieving significant speedup in the code is by performing the computations for BP only on the nodes that are connected to each other and going over the nodes and the edges only once. This is done in the function rowColumnIndex(). The function essentially collects, in arrays, the indices of the nodes that are connected to each other. For all remaining functions on the nodes, these arrays are used.

Once the constraints have been satisfied, the next step is to interpret the data; in the case of the max clique, it is to extract the clique. This is not strictly a post constraint satisfaction phase; the clique is extracted not after the constraints have been completely satisfied but as they are being satisfied in the outer loop. The clique-extractor works by sorting the nodes in ascending order of marginal probability of the node being included in the clique. The node with the highest probability is picked to be the starting node of the clique. The nodes with lesser probability are then tested if they are part of the clique or not. This is done by checking if the node in question is connected with all nodes already included in the clique. The clique extractor is set up in a way that it cannot pick an invalid clique. The extractor also keeps track of the number or skips, i.e. the number of nodes that have a high probability of being in the clique according to the BBN but are actually not part of the clique: This value is returned by the function. This number gives an indication if the BBN is actually doing some useful computation or applying some form of greedy or other approach to extract the clique.

Note that all the computations are done on the complement graph but the clique is extracted from the original graph. Performing the main computations on the complement graph, as explained above, makes it easier to work on dense graphs.

Implementation

The code was implemented in C++ on LINUX platform due to several reasons, but primarily due to the speed of computation provided in C++. The basic BBN code, i.e. the Node and Network classes have been kept as platform independent as possible. The compiler used was g++ version 2.95.

The arrays were implemented using pointers for dynamic allocation of memory. The system as of yet does not check if all the memory has been correctly allocated or not.

Parameter passing between the functions was avoided where the parameters being passed were the attributes of the class.

A lot of the complex computations of the network are reduced to simpler versions depending on the values of two key parameters: δ and β; Specifically when $\beta=1$ and $\delta=0$. The former led to a reduction of about 30% in execution time while the latter did not result in any significant reduction of execution time.

The BBN was tested on Second DIMACS Challenge graphs mentioned in the previous chapter. DIMACS provided two formats for the graphs: binary and ASCII [DIMA92].

```
c ASCII Graph Format of the DIMACS graphs for the second DIMACS
c implementation challenge, 1992
c the 'c' at the beginning of the line denotes the line is a comment
c The beginning of the problem data is marked by a line with 'p' as
c the first letter in the line.
c The same line for the clique problem would be something like:
p cliq 40 1245
c In the above line 'cliq' tells the graph is for the maximum clique
c problem. The number '40' is the number of vertices or nodes in the
c graph and '1245' is the number of edges in the graph.
c The edges of the graph are represented as 'e x1 x2'
c 'e' denotes that the line is an edge from node 'x1' to node 'x2'.
c There is only one edge specified per line
c so the following line denotes an edge from node 2 to 34
e 2 34
c The nodes are numbered from 1 to n where n is the number of
c vertices in the graph
```

5.3 ASCII graph format

Both the formats are supported by the BBN. The format of the ASCII graphs is given in figure 5.3. The ASCII format specifies the graph by listing all the edges of the graph. The binary format specifies the graph in a binary bitmap. The binary graph is more economical in terms of memory requirements but requires special functions to convert to and from the format. Both formats consist of a header preceding a listing of the graph. The format of the header is given in figure 5.4. The only difference between

the header formats for the ASCII and binary formats is that the latter is preceded by an integer number that gives the number if characters in the header. The statement starting with ‘p’ indicates it is the problem statement; it is followed by the word “cliq”--for the graphs of the max clique problem, and two numbers, indicating the number of nodes and edges in the graph respectively.

Graphs can be input to the BBN in either of the two formats or the BBN can randomly generate one with the required number of nodes and edge density. The random graphs are generated using a default seed of 32602. The user has the option of specifying the number of nodes, the edge density and the seed for graph generation. The user also has the option of saving the randomly generated graph in either the ASCII or the binary format. The network also provides the option of storing a binary input graph in ASCII format and vice versa. This allows converting a graph from ASCII to binary and binary to ASCII. It also allows the random graphs generated for testing the BBN to be stored in either format for future reference.

The DIMACS graphs start labeling their nodes from 1 while the BBN starts from 0. This is accounted for while reading the graph. When an input graph is provided in either format, the file is read and the data stored in the form of an adjacency matrix for the graph.

A final aspect of the implementation is the interface. The interface is currently specifically for the LINUX platform. The interface encapsulates the BBN classes and provides the options so the code can be run on different data and with different parameters without recompilation. A snapshot of the interface is provided in figure 5.4.

The format of the interface is the name of the parameter followed by the long option, the brief option and the default value of the variable. Any restrictions that are placed on the values that can be assigned to the variable are listed in parentheses next to the default value. An explanation of the options in the interface along with their explanation are listed below:

-n is the option to change the number of nodes in the network. This is for use only when a random graph is to be generated. If no value is specified, a graph of 20 nodes will be generated.

```
Bayesian Belief Network: Finding the Clique of a given Graph
Author: Amitoj Likhari, alikhari@cise.ufl.edu

The following command line options are available for this program
Description          Verbose Option   Brief  Default(restrictions)
Number of Nodes        --node           -n       20 (none)
Annihilate             --annihilate     -a       0.5 (<=1.0)
Beta                   --beta           -b       1 (!= 0)
Edge Probability       --edgeprob       -p       0.8 (<1.0)
Delta                  --delta          -d       0.0 (1.0)
Number of Outer Loops  --outer          -o       500 (none)
Number of Inner Loops  --inner          -i       1000 (none)
Constraint Threshold   --error          -e       1.0e-14 (none)
Integer Threshold      --intthresh      -t       0.05 (<1.0)
Seed                   --seed           -s       32602 (<32767)
Use input file         --inputfile      -f       should be in DIMACS
graph format
Save random graph      --savegraph      -g
Save graph ascii       --saveascii      -c
Save (out_nodes_seed)  --save           -v
```

5.4 Interface to the BBN

-a is the annihilate parameter. Higher values of annihilate push the values of the marginal probability towards an integer solution.

-b is the option to set the beta parameter in the network. Beta is one of the parameters of the MIME.

-e refers to the edge density. The default is set at 0.8, this means that the random graph generated will have an edge density of 0.8. Obviously, the upper limit on the value of the edge density is 1.0.

-d is for delta that is another parameter of the MIME.

-o and –i are self-explanatory; they set a limit on the number of outer and inner loop iterations respectively.

-e and –t are the thresholds for the inner and outer loop constraint satisfaction respectively. The threshold for the outer loop should always be less than 1.0 because it is a limit on the marginal probability.

-s is the seed for generating the random graph. The limit on its value is the range of values that integers can take in C++. This is mainly done to maintain portability over different platforms.

-f, -g, -c, and –v are options for specifying input and output files. –f specifies the file to be used for input, this file must be in the DIMACS ASCII or binary formats. –g specifies whether the random graph generated is to be saved. The graph is saved in a file named according to the seed used, the number of nodes in the graph and the edge density of the graph generated in the same order. The graph is saved in binary DIMACS format. The option –c specifies to save the input graph in DIMACS ASCII format. Minor adjustments need to be made to the code to provide the option of converting a graph from binary to DIMACS and vice versa without computing the clique on the graph. The final option, -v specifies whether to save the output for that particular graph. The format of the output file is explained below. A snapshot of an output file for the johnson32-4-4.clq.b

```
Nodes = 496 Edges = 107880
Delta = 1
Annihilate = 1
Beta = 1

      Final Clique - { 231 211 255 193 474 447 175 281 419 395 341 365 159

115 307 63 } - Size = 16 Iteration = 1

time in loops = 526.53
Clique in the first iteration
{ 435 466 408 395 9 367 350 20 299 252 35 209 54 168 77 104 } - Size = 16

Max Clique and iteration in which found
{ 231 211 255 193 474 447 175 281 419 395 341 365 159 115 307 63 } - Size =
16 Iteration = 1

 Iterations in which the clique size changed
Iteration = 1 Size = 16

Skips info
Iteration with skips > 0 and Number of skips in them
Iteration = 1 Number of Skips = 479
Iteration = 2 Number of Skips = 456
Iteration = 3 Number of Skips = 281
Iteration = 4 Number of Skips = 339
Iteration = 5 Number of Skips = 315
```

5.5 Format of the output file

file is given in figure 5.5. The file is saved with the name johnson32-4-4.clq.b.out.

The first few lines of the output file list information about the graph--number of nodes and number of edges--and the parameters of the BBN. It then lists the clique at the end of the required iterations of the outer loop along with the size and elements of the clique and the iteration in which it was found. It also lists the clique found in the very first iteration of the outer loop. The largest clique found by the BBN is listed next with the same information as for the clique in the first iteration. Finally, the number every iteration in which the number of skips made by the clique extractor changes is listed along with the number of skips made.

A Graphical User Interface (GUI) was implemented for the clique-BBN that supports all the features that the text version supports. The GUI was implemented in the Qt GUI toolkit by Trolltech, Inc. The toolkit has the advantage of being C++ compatible and can be executed on multiple platforms simply by recompiling the same code in on that platform. As of now, the GUI is still in a developmental stage and more work needs to be done before it can be released. The current version displays the graph in a regular polygon topology. The most likely clique at any point is displayed as the clique is being computed.

CHAPTER 6
RESULTS AND CONCLUSIONS

This chapter lists and explains the results of the BBN implementation. The testing strategy and details of the system on which testing was done are also given. The implications of these results are then discussed. The chapter also lists and explains some of the strong points of the system that can be exploited in future research and some extensions that can be made to the BBN to make it more robust, versatile and flexible. The chapter ends with a discussion about the possible applications of this particular formulation.

Testing

The BBN was tested on the benchmark graphs provided in the second DIMACS implementation challenge. The graphs are provided on the DIMACS site. The performance of the BBN was comparable to the benchmark programs provided in the same challenge. The results were unexpected because this is the first time BBN have been used to solve an NP Complete problem like the Maximum Clique. The BBN performance was either at par with or very close to the performance of the optimal program of the DIMACS challenge. The programs were tested with two different settings of the parameters on run on a AMD Athlon 1.3 GHz, 256 kb cache, 1 GB DDR 2100 DIMM running Redhat Linux 7.3., kernel 2.4.18. The major advantage of the MIME-BBN approach over the other approaches to the same problem is the speed of computation over dense graphs. This is because it follows the complement graph

approach explained in the previous chapter; the complement graph for a dense original graph is very sparse and so the number of computations reduces with increase in density of the graph. This advantage is not really apparent from comparing the results on the DIMACS benchmarks because the DIMACS benchmark graphs are not very dense; the maximum density being about 0.75.

The testing strategy was to test the BBN on the DIMACS challenge graphs and the randomly generated graphs. Testing was done using a script file with the parameters hardwired in the script file. The benchmark graphs were run alphabetically. The BBN was run for only 5 iterations of the outer loop, arguably the performance can increase if the BBN is run for a longer time.

It could be argued that the BBN rank orders the nodes according to some greedy criterion. Although possible, this is quite improbable. The clique extractor also counts the total number of nodes skipped in order to compute the clique. This number does not always decrease, as it would in a greedy approach. The number of skips actually increases at times without a change in the size of the clique extracted.

The BBN was run on two different values for the annihilation parameter. The parameter forces the BBN to go to an integer solution. The results were apparent only one of the graphs where the one with a higher value of the annihilation parameter resulted in a larger clique in the same number of iterations than the one with a smaller value of the parameter.

Results

A few of the results are listed here in figure 6.1 followed by a discussion on the same. A detailed listing of the results is provided in Appendix A. A discussion of the

results is provided in the next section along with the some improvements that can be made to the code.

Name	Size	Edges	BBN Clique Found	Optimal Clique
Brock200_1	200	14834	16	21
C-Fat200-1	200	1534	12	12
C-Fat500-1	500	23191	62	62
Hamming8-2	256	31616	128	128
Hamming10-2	1024	518656	496	512
Hamming10-4	1024	434176	31	40[1]
Johnson32-4-4	496	107880	16	16
P_Hat300-2	300	21928	23	25
San200_0.9_1	200	17190	45	70
San400_0.9_1	400	71820	42	100

6.1 Summary of results

Discussion of Results

The above results represent the gamut of results obtained from the testing. The BBN comes up with very bad cliques in very few cases. In general, the performance on the cliques of the San family of benchmarks were the ones on which the BBN performed the worst and failed to find cliques of sizes comparable to the optimal clique size. In general, the clique size found by the BBN for this family was half of the optimal clique size found. The cliques for the P_Hat family are mixed giving close to optimal in some cases and little less for others. The best results were obtained for the C-Fat and Hamming families of graphs. On these, the BBN found optimal or close to optimal results even on very large graphs. The largest graphs (500 and 1024 nodes for P-Fat and Hamming) that were run on the BBN belong to this family. Finally the Brock graphs resulted in cliques whose sizes were always less than optimal but still close to optimal.

Future research

The thesis is the first known application to the BBN to the field of NP complete combinatorial optimization problems. The results of applying the BBN to the clique have been highly encouraging.

There are many things that can and must be done to explore the area completely. One of the first things is to let the BBN run for more than the five iterations of the outer loop that it was executed for in the current work. This could possibly result in better results to the clique problem. The BBN should also be run on the benchmarks with different values of the parameters. In the current work, the BBN was run mainly for one value of 'beta' and 'delta' parameters. The function of the annihilate parameter also needs to be explored, the results obtained in this thesis were for 'annihilate' value of 0.5 and 1.0. The BBN needs to be run with more values of annihilate in the range 0.0 to 1.0.

A second area where the algorithm could arguably be made better is the values of the joint probability. The algorithm currently looks at only one of the joint probability values.

The MIME BBN essentially results in a rank ordering of the nodes according to the probability of their being included in the maximal clique. It might also be worthwhile to look extract not the most likely clique but the second or third cliques. That is, the clique to be extracted would start from not the first node in the rank ordering but the first node that is not part of the clique, which the most probable node is a part of.

One of the most important areas of improvement may be in the free energy function used. Currently, the MIME BBN uses the Bethe free energy that considers the pairwise link functions; this means the algorithm uses pairwise constraint satisfaction. A

logical direction to proceed would be to use the Kikuchi free energy that can be used for triple or higher constraint satisfaction. This would make the algorithm slower but may possibly result in much better solutions.

Applications

The thesis applies a brand new approach to a combinatorial optimization problem. The approach is BBN and the problem is the Maximum Clique Problem. Therefore the application areas of the thesis are the domains where either of the two problems applies. The most application would be in the area of computer vision.

Computer Vision consists essentially of graph matching. Where there is an unknown shape that has to be matched to a template and classified accordingly. Examples of the shape and the object can be frontal view of a bus being matched to a diagonal frontal view and the algorithm has to come up with the matching. So the shape has to be transformed to match the template by some translation or rotation etc. A common approach to graph matching is to transform the graph into an auxiliary graph structure known as an association graph and finding the clique in that.

Most of the more popular current algorithms for this are based on the Motzkin-Strauss [MOTZ65] theorem to provide a formulation for the maximal clique as a quadratic programming problem. One algorithm to solve this is based on payoff-monotonic dynamics [PELL02]. The formulation the above algorithm is very similar to that the MIME is based on for the clique. It has been found that such continuous solutions to discrete problems are more helpful, especially in trying to come up with the parameters of translation and rotation to transform the shape to the template. The MIME BBN excels in this area, the result of the algorithm is not a hard matching between points as in other traditional heuristics but it gives probabilities with which a point in the shape

matches with the template. This adds to the robustness of the actual computer vision algorithm.

Bayesian networks have only recently been applied to this area and with an application to the maximum clique, which is also applicable to the same problem, the MIME BBN might prove to be a valuable tool.

APPENDIX
RESULTS OF THE BBN ON DIMACS GRAPHS

The appendix lists the comprehensive results of the running the BBN on the DIMACS benchmark graphs. The results for the benchmark approaches are listed on the DIMACS ftp site for the second implementation challenge (ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/results/dmclique). The best results obtained from the benchmark programs are also listed. There are two benchmark programs: the dfmax.c, which gives the optimal clique and the dfclique.c that is a semi exhaustive greedy heuristic. Some of the programs were run only on the dfclique.c, that is indicated where applicable. Column 1 lists the name of the graph, column 2 and 3 specify the number of nodes and edges respectively, and the maximum clique is listed in column 4. The best results obtained by the Bayesian belief Network (best of all settings) and the benchmark approaches are listed in columns 5 and 6.

Name	Nodes	Edges	Optimal Clique	BBN Clique	Bench Mark Clique
Brock200_1	200	14834	21	16	21
Brock200_2	200	9876	12	7	12
Brock200_3	200	12048	15	12	15
Brock200_4	200	13089	17	11	17
Brock400_1	400	59723	27	19	24[1]
Brock400_2	400	59786	29	20	25[11]
Brock400_3	400	59681	31	20	25[1]
Brock400_4	400	59765	33	19	25[1]
C-FAT200-1	200	1534	12	12	12
C-FAT200-2	200	3235	24	23	24
C-FAT200-5	200	8473	58	58	58
C-FAT500-1	500	4459	14	14	14
C-FAT500-2	500	9139	26	26	26
C-FAT500-5	500	23191	64	62	64
C-FAT500-10	500	46627	??	126	126[1]
Hamming6-2	64	1824	32	32	32
Hamming6-4	64	704	4	4	4
Hamming8-2	256	31616	128	128	128
Hamming8-4	256	20864	16	12	16
Hamming10-2	1024	518656	??	496	512[1]
Hamming10-4	1024	434176	40	31	40
Johnson16-2-4	120	5460	8	8	8
Johnson8-2-4	28	210	4	4	4
Johnson8-4-4	70	1855	14	11	14

[1] The graphs indicated were run on the heuristic benchmark program (dfclique.c) and not on the optimal exact algorithm.

Johnson32-4-4	496	107880	16	16	16
Keller4	171	9435	11	7	11
Name	Nodes	Edges	Optimal Clique	BBN Clique	Bench Mark Clique
Keller 5	776	225990	27	15	27
P_HAT300-1	300	10933	8	5	8
P_HAT300-2	300	21928	25	23	25
P_HAT300-3	300	33390	36	31	36
P_hat500-1	500	31569	9	5	9
P_hat500-2	500	62946	36	23	36
P_hat500-3	500	93800	??	42	49
San200_0.7_2	200	13930	18	14	18[1]
San200_0.7_3	200	17863		36	
San200_0.9_1	200	17910	70	45	70[1]
San200_0.9_2	200	17910	60	37	60[1]
San200_0.9_3	200	17910	42	31	42[1]
San400_0.5_1	400	39900	13	7	13[1]
San400_0.5_2	400	55860	??	21	??
San400_0.7_1	400	55860	40	21	40[1]
San400_0.7_2	400	55860	30	16	30[1]
San400_0.7_3	400	55860	22	13	22[1]
San400_0.9_1	400	71820	100	42	100[1]
Sanr200_0.7	200	13868	18	14	18[1]
Sanr200_0.9	200	17863	42	36	42
Sanr400_0.5	400	39984	13	12	13
Sanr400_0.7	400	55869		16	

LIST OF REFERENCES

[AKKO73] Akkoyunlu, E.A., The enumeration of maximal cliques of large graphs, *SIAM Journal on Computing*, 2: 1--6, 1973.

[AUGU70] Auguston, J.G., and Minker, J., An analysis of some graph theoretical cluster techniques, *Journal of Applied Computation and Mathematics*,17(4):571--588, 1970.

[BOMZ99] Bomze, I.M., Budinich, M., Pardalos, P.M., Pellilo, M., The maximum clique problem, *Handbook of Combinatorial Optimization (Supplement Volume A)*, Du, D.Z., and Pardalos, P. M., (Eds.), Kluwer Academic Publishers, Boston, MA, 1999.

[BOPP92] Boppana, R., and Haldorsson, M.M., Approximating maximum independent sets by excluding subgraphs, *Bit*, 32: 180--196, 1992.

[CARR90] Carraghan, R., and Panos, P., An exact algorithm for maximum clique, *Operations Research Letters*, 9: 375--382, 1990.

[COOP90] Cooper, G., Computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence*, 42:393--405, 1990.

[CORM01] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., *Introduction to Algorithms*, 2nd edition, MIT Press, Cambridge, MA, 2001.

[COZM00] Cozman, F.G., Generalizing variable elimination in Bayesian networks, *Proceedings of the Workshop on Probabilistic Reasoning in Artificial Intelligence*, 27--32, 2000.

[DAGU93] Dagum, P., and Luby, M., Approximating probabilistic inference in Bayesian belief networks is NP-hard, *Artificial Intelligence*, 60: 141--153, 1993.

[DAGU94] Dagum, P. and Luby, M. On the approximation of probabilistic inference. *Technical Report, section on Medical Informatics, Stanford University School of Medicine,*1994.

[DEAN95] Dean, T., Allen, J., and, Aloimonos, Y., *Artificial Intelligence: Theory and Practice*, Benjamin/Cummings Publishing Company, Redwood City, CA, 1995.

[DIMA92] ftp site of the second DIMACS challenge on NP-complete problems, 1992. *ftp://dimacs.rutgers.edu/pub/challenge/graph/doc/ccformat.tex*, 1992, 6/23/02.

[FEOT95] Feo, T.J and Resende, M.G., Greedy randomized adaptive search procedures: *Journal of Global Optimization*, 6: 109-133, 1995.

[GUOH02] Guo, H., and Hsu, W, A survey for real-time Bayesian network inference, *http://www.kddresearch.org/Workshops/RTDSDS-2002/papers/RTDSDS2002-GH-01.pdf,* 2002, 10/06/02.

[HARA57] Harar, F., and Ross, L.C., A Procedure for clique detection using the group matrix, *Sociometry*, 20: 205-215, 1957.

[HECK95] Heckerman, D., A tutorial on learning with Bayesian networks, *Technical Report MSR-TR-95-06,* Microsoft Research, Seattle, WA, 1995.

[HECK96] Heckerman, D., A tutorial on learning with Bayesian networks, *Technical Report MSR-TR-9506,* Microsoft Research, Seattle, WA, 1996.

[HECK98] Heckermann, D., Sahami, M., Dumais, S., and Horovitz, E.,. A Bayesian approach to filtering junk e-mail. *AAAI'98 Workshop on Learning for Text Categorization,* 1998.

[HORO98] Horovitz, E., Sahni, S., and Rajasekeran, S., *Computer Algorithms*, W. H. Freeman, New York, NY, 1998.

[HOWA81] Howard, R., and Matheson, J., Influence Diagrams, in *Readings on the Principles and Applications of Decision Analysis, volume II* (Eds. Howard, R. and Matheson, J.), Strategic Decisions Group, Menlo Park, CA, 1981.

[JAGO95] Jagota, A., Approximating maximum clique with a Hopfield network, *IEEE transactions in Neural Networks*, 6: 724-735, 1995.

[JAGO96] Jagota, A., Sanches, L., Ganesan, R., Approximately solving the maximum clique using neural networks and related heuristics, in *[JOHN96]*.

[JOHN96] Johnson, D.S. and Trick, M.A., (Eds.) *Volume 26.: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1992,* American Mathematical Society, Providence, RI, 1996.

[JORD97] Jordan, M.I., An introduction to graphical models (Lecture notes), *http://www.ai.mit.edu/projects/jordan.html*, 1997, 08/04/02.

[KARP72] Karp, R.M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Miller, R.E. and Thatcher, J.W. (Eds.), Plenum Press, New York, 1972.

[KIRK83] Kirkpatrick S., Gelatt, C.D., and Vecchi, M.P., Optimization by simulated annealing, *Management Science*, 22: 671--680, 1983.

[KSCH00] Kschischang, F.R., Frey, B.J., Loeliger, H., Factor Graphs and the sum product algorithm. (Submitted to IEEE Transactions on information theory, 1998, revised 2000), *http://www.comm.utoronto.edu.ca/frank/factor,* 11/03/02.

[LAUR88] Lauritzen, S.L., and Spiegelhalter, D.J., Local computations with probabilities on graphical structures and their application to expert systems, *Journal of Royal Statistical Society*, 50:157--224, 1988.

[MICR02] Microsoft Research Decision Theoretic Group, Decision theory and adaptive systems, *http://www.research.microsoft.com/research.dtg/*, 2002, 10/12/02.

[MITC97] Mitchell, T.M., *Machine Learning*, McGraw Hill Publishing Company, New York, NY, 1997.

[MURP99] Murphy, K.P., Weiss Y., and Jordan, M.I., Loopy belief-propagation for approximate inference: an empirical study. In *Proceedings of Uncertainty in AI*, 9: 467--475, 1999.

[MURP01a] Murphy, K.P., A brief introduction to graphical models and Bayesian networks, *http://www.cs.brkeley.edu/~murphy/Bayes/bayes.html*, 2001, 08/10/01.

[MURP01b] Murphy, K.P., The Bayes Net Toolbox for MATLAB, *http://citeseer.nj.nec.com/murphy01bayes.html*, 2001, 10/15/02.

[OLMS83] Olmsted, S., *On Representing and Solving Decision Problems*, PhD Thesis, Department of Engineering-Economic Systems, Stanford University, 1983.

[OSTE02] Östergård, P.R.J., A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics,* 120:195--205, 2002.

[PARD99] Pardalos, P., Rappe, J., and Resende, M.G.C. An exact parallel algorithm for the maximum clique problem, in *High Performance Algorithms and Software in Nonlinear Optimization,* R. De Leone, A. Murl'i, P.M. Pardalos and G. Toraldo (Eds.), Kluwer Academic Publishers, Boston, MA, 1999.

[PEAR88] Pearl, J., *Probabilistic reasoning in intelligent systems: networks of plausible inference,* Morgan Kaufmann Publishers, San Mateo, CA, 1988**.**

[PEAR00] Pearl, J., and Russel, S., Bayesian networks, *Technical Report, R-277,* Cognitive Systems Laboratory, University of California, Los Angeles, 2000.

[PELI02] Pelillo, M., Matching free trees, maximal cliques, and monotone game dynamics, *IEEE transactions on Pattern Analysis and Machine Intelligence,* 24(11): 1535--1541, 2002.

[RANG02] Rangarajan, A. and Yuille, A.L., MIME: Mutual information minimization and entropy maximization for Bayesian belief propagation, in *Advances in Neural Information Processing Systems 14,* Dietterich, T.G., Becker, S., Ghahramani, Z., (Eds.), MIT Press, Campbridge, MA, 2002.

[SCHA88] Scahchter, R., Probabilistic Inference and Influence Diagrams, *Operations Research*, 36: 589--604, 1988.

[SPIE86] Spiegelhalter, D.J., Probabilistic reasoning in predictive expert systems, in *Uncertainty in Artificial Intelligence*, Kanal, L.N., and Lemmer, J.F., (Eds.): 47--68, 1986.

[VAND96] van der Gaag, L.C., Bayesian belief networks: odds and ends, *The Computer Journal*, 39: 97--113, 1996.

[WOOD97] Wood, D.R., An Algorithm for Finding the Maximum Clique in a Graph, *Operations Research Letters*, 21(5):211--217, 1997.

[YEDI01] Yedidia, J.S., Freeman, W.T., and Weiss, Y., Generalized belief propagation, in *Advances in Neural Information Processing Systems 13*, Leen, T.K., Dietterich, T.G., amd Tresp, W., (Eds.), MIT Press, Cambridge, MA, 2001.

[YUIL01] Yuille, A.L., A double loop algorithm to minimize the Bethe and Kikuchi free energies, *www.cs.utoronto.edu/~mackay/bethe.ps.gz*, 2001**.** 06/12/02**.**

BIOGRAPHICAL SKETCH

I graduated with a Bachelor of Technology in computer science and engineering from Guru Nanak Dev University, Amritsar, India, in 1999. I love playing basketball and led my university team to second place in the country during my senior team. My interests include artificial intelligence, reasoning under uncertainty, and robotics. I started my master's in computer engineering at the University of Florida in fall, 2000. I am currently working towards a master's in exercise and sports science with a concentration in motor learning and control at the University of Florida.