Dynamic Weighting A^* Search-based MAP Algorithm for Bayesian Networks[1]

Xiaoxun Sun
Department of Media & Knowledge Engineering
Delft University of Technology
The Netherlands
{x.sun@ewi.tudelft.nl }

July 8, 2005

[1] Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Media & Knowledge Engineering, Delft University of Technology, The Netherlands

Graduate Committee

of Xiaoxun Sun (TU Delft student No. 1194771)

Dr. Drs. L.J.M. Rothkrantz
Dr. A.H.J.Oomes
Dr. K. van der Meer
Dr. Ir. Marek J.Druzdzel

Abstract

Maximum a *Posteriori* assignment (MAP) is the most probable instantiation of a set of variables given a partial evidence on the remaining variables in a Bayesian network. Finding MAP has been proven to be an NP-hard problem [20], and it is not only exponential in the network treewidth, but also in the *constrained treewidth* [13]. Exact approaches often fail to yield any results for MAP problems in very large Bayesian networks, and even approximate approaches may not yield acceptable solutions.

We introduce the Dynamic Weighting A^* (DWA^*) search algorithm for solving MAP. By exploiting asymmetries in the distribution of MAP variables, the algorithm is able to greatly reduce the search space, yielding very good quality MAP solutions. Experimental results demonstrate that my algorithm finds solutions generally faster and with a lower variance in search time than existing algorithms.

Contents

Acknowledgements

The fantastic time I enjoyed at the University of Pittsburgh, could not have been possible without the support of all the people involved in my graduation project and my stay in Pittsburgh.

First and foremost, I would like to thank my advisor from the Delft University of Technology, Drs. Dr. L.J.M. Rothkrantz and my supervisor at the University of Pittsburgh, Dr. Ir. Marek J. Druzdzel. Not only did their instructions lead me to realize the excitement of research, but also their strong support and continuous effort endowed me with power when facing challenges on the way. I could have never taken directed steps to prepare myself for a career in science without their inspiration.

Next, I would like to thank my fellow colleagues at the Decision Systems Laboratory for their support and help. In particular, Changhe Yuan and Adam Zagorecki for always being close friends and offering insightful comments on my research.

My life and study in the Delft University of Technology could not have been so exciting without my Dutch classmates and close friends: Paul Klapwijk, Dennis Joele, and Vincent de Lange.

Last but certainly not the least, I would like to express my sincere gratitude to Xiaotian Shen for her love, no matter how far away I was.

This research was supported by the Air Force Office of Scientific Research grant F49620–03–1–0187. I thank Adam Zagorecki and Tomek Sowinski for insightful comments that led to improvements in the thesis. I thank James D. Park for answering the question regarding his systematic search algorithm and I thank Adnan Darwiche and Keith Cascio for providing us with the latest version of the P-Sys and P-Loc algorithms within the SamIam software. All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory and available at `http://www.sis.pitt.edu/~genie`.

Chapter 1

Introduction

The purpose of this thesis is to describe the research I carried out in the Decision Systems Laboratory (DSL) at the School of Information Sciences of the University of Pittsburgh. In short, the main objective of this research is to develop an efficient and accurate algorithm for solving the Maximum a *Posteriori* Assignment (MAP) problem in Bayesian networks.

1.1 Background Introduction

The Maximum a *Posteriori* assignment (MAP) is the problem of finding the most probable instantiation of a set of variables given partial evidence on the remaining variables in a Bayesian networks. A Bayesian network [17] (also known as a belief network or probabilistic network) is a formalism for reasoning under uncertainty. Decision support based on probabilistic reasoning was developed in the late 1970s and gained popularity when efficient algorithms for inference were introduced in Bayesian networks [12]. Thanks to an intuitive graphical interface and a sound probabilistic framework, the Bayesian network has become a popular approach to model various expert systems, e.g., medical, image interpretation, troubleshooting, and information processing.

In detail, a Bayesian network is an acyclic directed graph that represents a factorization of the joint probability distribution over a set of random variables.

The graphical structure of the network is the qualitative part of a Bayesian network and embodies a set of nodes representing the random variables and a set of arrows representing direct dependencies between connected variables. Absence of an arrow between variables implies that these variables are (conditionally) independent. The parents of a variable are the variables which are connected with an arrow with its direction going into this variable.

The joint probability distribution is the quantitative part of a Bayesian network and embodies the conditional probability distribution defined with each

variable. This distribution characterizes the influence of the values of the predecessors (parents) on the probabilities of the values of the variable itself. When a variable has no parents, the probability distribution is the prior probability distribution. In practice, these distributions are derived from frequency data or elicited from an expert judgment.

Given a joint probability distribution over a set of random variables, many different graphs exist which factorize the same joint probability distribution. A factorization that is especially desired is the graph that reflects the causal structure of the problem. This graph, also known as a causal graph, normally reflects an expert's understanding of the domain and facilitates a user's insight during the operational stage.

One specialization of the MAP that has been paid much attention is the Most Probable Explanation (MPE) problem. MPE is the problem of finding the most probable assignment of a set of variables given *full* evidence of the remaining variables. MAP turns out to be a very difficult problem even when compared to MPE or computing the probability of evidence. Particularly, the decision problem for MPE is NP-complete while the corresponding MAP problem is NP^{PP}-complete [13]. MAP is more useful than MPE for providing explanations. For instance, in diagnosis, generally we are only interested in the configuration of fault variables given some observations. There may be many other variables that have not been observed and are outside the scope of our interest.

The formula to compute the probability of each possible scenario of MAP is not too complex. Give a Bayesian network, let **M** be the set of MAP variables, the configuration of which is what we are interested in; **E** is the set of evidence, namely the variables whose states we have known; The remainder of the variables, denoted by **S**, are variables that we neither know their states nor care about their configuration. If a variable in the set of MAP variables **M** is intantiated at the ith place using its jth state, it will be denoted as $\mathbf{M}_{ij}$.

By using chain rule, the probability of the MAP problem which consists of **n** MAP variables can be presented as follows:

$$\begin{aligned} P(\mathrm{M} \mid E) &= P(M_{ni} \mid M_{1j}, M_{2k}, \ldots M_{(n-1)t}, E) \\ &\quad \ldots P(M_{2k} \mid M_{1j}, E) P(M_{1j} \mid E) \ . \end{aligned}$$

Each posteriori probability at the righthand side of the equation above can be computed by the jointree algorithm [12] efficiently. In other words, the MAP problem is to find the scenario with the largest posteriori probability among all possible assignments to the **M** given **E**.

1.2 Motivation and Objective

Several researchers have proposed algorithms for solving the MAP problem. A very efficient approximate search-based algorithm based on local search, pro-

posed by Park and Darwiche [13], is capable of solving MAP efficiently which is based on local search. An exact method, based on branch-and-bound depth-first search, proposed by Park and Darwiche [15], performs quite well when the search space is not too large. Another approximate proposed more recently by Yuan et al. [21] is a Reheated Annealing MAP algorithm. It is somewhat slower on simple networks but it is able to handle very hard cases which the exact algorithm can not solve.

In my thesis, I propose the Dynamic Weighting A^* (DWA^*) Search algorithm for solving MAP that is faster than any of the existing algorithms. The algorithm explores the asymmetries among all possible assignments in the joint probability distributions. Typically, a small fraction of assignments can be expected to cover a large portion of the total probability space with the remaining assembles having practically negligible probability [7].

Previous research and simulation results have shown that the *greedy guess* [14, 21], which is represented as follows:

$$P(M|E) = \prod_{i=1}^{n} \max_{j} P(M_{ij}|M_{(i-1)k} \ldots M_{1m}, E) \tag{1.1}$$

is quite close to the optimal solution of the MAP problems. In other words, it offers a very tight lower bound on the optimal solution. While it is theoretically not admissible (admissible heuristic should offer an upper bound on the MAP), with a simple extension it offers ϵ-admissibility [16] and excellent performance.

1.3 Overview

The remainder of this thesis is structured as follows. Section 2 defines the MAP problem and summarizes the main results on its complexity. It also outlines several methods for solving MAP. Section 3 introduces the theory of asymmetries among joint probability distributions. Section 4 describes the Dynamic Weighting A^* Search algorithm. Section 5 describes the implementation of the algorithm. Section 6 presents the results of applying the algorithm to several real complex Bayesian networks.

Chapter 2

MAP and Previous Research

2.1 Bayesian Networks

This section presents a brief introduction into Bayesian networks and describes the necessary concepts for this thesis. I assume that the reader is familiar with the essentials of theory and probability theory.

A Bayesian network [17] (also known as a belief network or probabilistic network) is a formalism for reasoning under uncertainty. Decision support based on probabilistic reasoning was developed in the late 1970s and gained popularity when efficient algorithms for inference were introduced in Bayesian networks [12]. Thanks to an intuitive graphical interface and a sound probabilistic framework, the Bayesian network has become a popular approach to model various expert systems, e.g., medical, image interpretation, troubleshooting, and information processing.

In detail, a Bayesian network is an acyclic directed graph that represents a factorization of the joint probability distribution over a set of random variables.

The graphical structure of the network is the qualitative part of a Bayesian network and embodies a set of nodes representing the random variables and a set of arrows representing direct dependencies between connected variables. Absence of an arrow between variables implies that these variables are (conditionally) independent. The parents of a variable are the variables which are connected with an arrow with its direction going into this variable.

The joint probability distribution is the quantitative part of a Bayesian network and embodies the conditional probability distribution defined with each variable. This distribution characterizes the influence of the values of the predecessors (parents) on the probabilities of the values of the variable itself. When a variable has no parents, the probability distribution is the prior probability

distribution. In practice, these distributions are derived from frequency data or elicited from an expert judgment.

Given a joint probability distribution over a set of random variables, many different graphs exist which factorize the same joint probability distribution. A factorization that is especially desired is the graph that reflects the causal structure of the problem. This graph, also known as a causal graph, normally reflects an expert's understanding of the domain and facilitates a user's insight during the operational stage.

Example 1. Consider the Bayesian network in Figure 2.1, which represents a fictitious Asia example from Spiegelhalter and Knill-Jones [1984]. This network is based on the knowledge that dyspnea (**DY**), i.e., shortness-of-breath, may be due to tuberculosis (**TC**), lung cancer (**LC**), or bronchitis (**BC**). A recent visit to Asia (**VA**) increases the probability of tuberculosis, while smoking (**SM**) is known to be a risk factor for both lung cancer and bronchitis. Neither the result of a single chest X-ray (bf XR) nor the presence or absence of dyspnea, discriminates between lung cancer and tuberculosis. Each of the variables is associated with a probability distribution. So has the variable **SM** the marginal probability distribution of Table 2.1. And, since the variable **SM** is the parent of the variable **LC**, this variable has a conditional probability distribution of **LC** conditioned on **SM**, see Table 2.2.

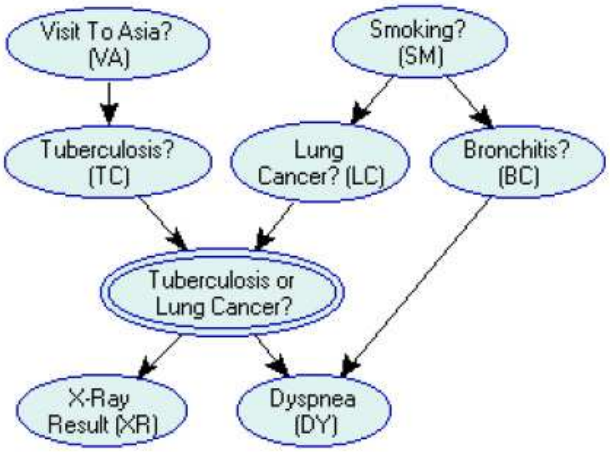

Figure 2.1: An example of Bayesian network.

The **jointree** algorithm, a various efficient algorithms first proposed by Lauritzen and Spiegelhalter [12] exists for reasoning in Bayesian networks, e.g., determining the impact of processing evidence into the network. Although the

Table 2.1: Prior probability table of the variable SM.

	Pr(SM)
SM_nonsmoker	0.75
SM_smoker	0.25

Table 2.2: Conditional probability table of the variable LC conditioned on the variable SM.

Pr (LC| SM)	SM_nonsmoker	SM_smoker
LC_absent	0.75	0.45
LC_present	0.25	0.55

calculation of probabilistic inference is NP-hard, the algorithms provide reasonable computing times for networks consisting of tens or even hundreds of nodes.

Before I present the definition of a Bayesian network and Bayes rule, I introduce some necessary notations. Consider a finite set of discrete random variables V, where each variable X ∈ V is denoted as a capital letter, e.g., X, Y, Z. Each state of a variable is denoted as a lowercase letter, e.g., x, y, z. The set of all states within a variable X, is denoted as DX. The probability distribution over a random variable X is denoted as Pr(X) and the probability of a state x ∈ DX as Pr(X = x) or in shorter form Pr(x).

A combination of states of multiple variables is denoted as a scenario. The set of all the scenarios from a set of variables V, is denoted as D_V, and each scenario as s ∈ D_V. In case of one variable, the set of scenarios and the set of states of the variable are identical. In Table 2.2 from Example 1 the variables LC and SM yield the four scenarios displayed in Table 2.3. The probability of a scenario is defined by the joint probability over the states in the scenario. The probability distribution over a set of variables is denoted as Pr(V) and the probability of a scenario s ∈ DV as Pr(V = s) or in shorter form Pr(s). The set of parents of a variable X is denoted as $\prod X$.

Table 2.3: Four possible scenarios of the variables SM and LC.

SM_nonsmoker & LC_absent	SM_nonsmoker & LC_present
SM_smoker & LC_absent	SM_smoker & LC_present

The foundation of the Bayesian network is the Bayes theorem,

$$Pr(B \mid A) \quad = \quad \frac{Pr(A \mid B)Pr(B)}{Pr(A)}.$$

named after Reverent Thomas Bayes (1702-1761). The initial probability Pr(A) is called the prior probability, and the updated probability $Pr(A \mid B)$ the

posterior probability. An interpretation of the posterior probability is the probability of A with the knowledge of the state of variable B. When the knowledge of a variables has an effect on the probability of another variable these variables are called dependent. If variables are independent of each other, the posterior probability and the prior probability are equal, $Pr(A \mid B) = Pr(A)$.

Definition 1 Bayesian network. *A Bayesian network, BN = $\langle G, \Theta \rangle$ is an acyclic directed graph, $G = \langle \mathcal{V}, \mathcal{A} \rangle$, where the arrows A denote a probabilistic relation between the vertices and each vertex, $V \in \mathcal{V}$ represents a discrete random variable. Associated with the vertexes is a $\theta_{V \in \mathcal{V}} : D_V * D_{\Pi_V} \rightarrow$ [0, 1] function with the condition that for each combination of $\pi_V \in \Pi_V$, there holds:*

$$\sum_{dV \in D_V} \theta_V(d_V, \pi_V) \quad = \quad 1.$$

The probability distribution of each variable is embodied by the joint probability distribution encoded in a Bayesian network. Suppose for example two variables, A and B, with the joint probability distribution Pr(A,B). With marginalization, the probability distribution of A is calculated by taking the sum over the joint probability of A with all the states of B.

$$Pr(A) \quad = \quad \sum_{b_i \in D_B} Pr(A, b_i)$$

In order to determine and present the joint probability, the following theorem better known as the chain rule may be applied.

Definition 2 Chain rule. *Let BN be a Bayesian network over a finite set of discrete random variables $\mathcal{V} = \{V_1, \ldots, V_n\}$. The joint probability distribution $Pr(\mathcal{V})$ is then,*

$$Pr(\mathcal{V}) \quad = \quad \sum_{i=1}^{n} Pr(V_i \mid \Pi_{V_i}).$$

When variables are instantiated (=set to a state) I refer to these variables as evidence. A possible effect of entering evidence is a change in the dependency relations between variables, i.e., different variables may become independent of or dependent on each other. When two sets of variables become independent of each other given the instantiation of a third set, this is identified as conditional independence.

Definition 3 Conditional independence. *Let V be a finite set of discrete random variables and let Pr (V) denote the joint probability distribution over the variables. Suppose three disjoint subsets of variables, $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \subset \mathcal{V}$. The sets*

$\mathcal{X}$ and $\mathcal{Y}$ are conditionally independent given $\mathcal{Z}$, if for all $s_x \in D_X$, $s_y \in D_Y$, and $s_z \in D_Z$, thereholds :

$$Pr(s_x \mid s_y, s_z) \quad = \quad Pr(s_x \mid s_z).$$

Definition 4 d-separation. *Let BN be a Bayesian network over a finite set of discrete random variables V and let X, Y, and Z stand for any three disjoint subsets of variables of V. Z is said to d-separate X from Y, if along every path (sequence of connected variables) between a variable in X and a variable in Y, there is a variable W satisfying one of the following two conditions: (1) W has converging arrows and none of W or its descendants are in Z, or (2) W does not have converging arrows and W is in Z. The sound mathematical framework and the support for conditional independence and d-separation make a BN a powerful tool for modelling probability relations between random variables.*

2.2 MAP

Conceptually, give a Bayesian network, the MAP problem is defined as follows. Let **M** be the set of MAP variables, the configuration of which is what we are interested in; **E** is the set of evidence, namely the variables whose states we have known; The remainder of the variables, denoted by **S**, are variables that we neither know their states nor care about their configuration. Given an assignment e of variables **E**, the MAP problem is that of finding the assignment **m** of variables **M** which maximizes the probability of $P(m \mid e)$, while the MPE problem is the special case of MAP, when **S** is empty.

$$map = \max_{M} \sum_{S} p(M, S \mid E) \; . \tag{2.1}$$

In general, in Bayesian networks, we use the Conditional Probability Table (CPT) ϕ as the *potential* over a variable and its parent nodes. A potential over all the states of one variable after updating beliefs is called *marginal*. The notation ϕ_e stands for the potential in which we have fixed the value of $e \in E$.

Then the probability of MAP with Φ as its CPTs turns out to be a real number:

$$map = \max_{M} \sum_{S} \prod_{\phi \in \Phi} \phi_e \; . \tag{2.2}$$

We will introduce the algorithm of *Variable Elimination* [15] here in order to compute MAP. The name of the algorithm is just because it sums or maximizes out variables from a list of variables one by one, and this order is named the *elimination order*. The size of the largest *clique* [12] minus 1 in a *join-tree* constructed based on an elimination order is called the *induced width*. The

induced width of the best elimination order is called the *treewidth*. In computing posterior marginal distributions, we only have summations. Thus, we can commute summations over different variables in order to minimize the induced width of an elimination order. Similarly, we have only maximizations in an MPE problem. Once again, any permutation of the maximizations over different variables is admissible. Hence, the above two problems can be solved using treewidths. However, a MAP problem has both maximizations and summations. Since summation and maximization do not commute, we are required to do summations first. An elimination order is *valid* if maximizing a variable out of a potential never happens before summing over another variable on the same potential [13]. The induced width of the best elimination order under certain constraints is called the *constrained width*. Because of the inherent constraints that MAP problems enforce on elimination orders, they are subject to the constrained widths of the best valid elimination orders.

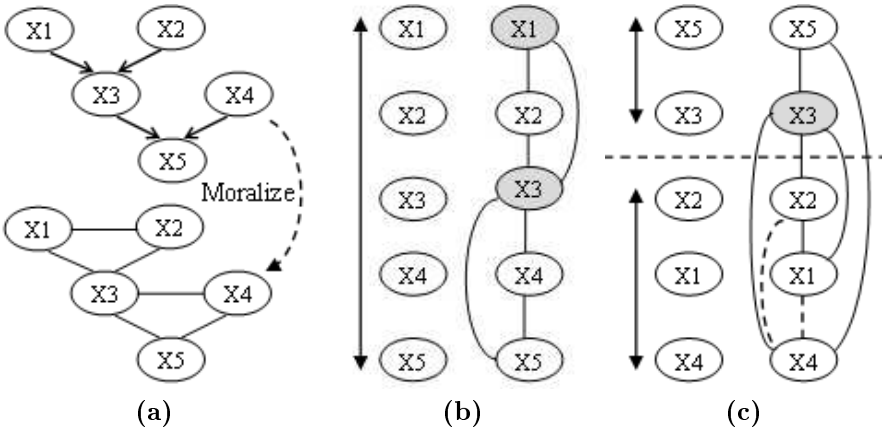

Figure 2.2: **(a)** A simple Bayesian network and its moralized graph; **(b)** The induced graph for solving its MPE problem: $\max_{X_1,X_2,X_3,X_4,X_5} P(X_1, X_2, X_3, X_4, X_5)$; **(c)** The induced graph for solving the following MAP problem: $\max_{X_1,X_2,X_4} \sum_{X_3,X_5} P(X_1, X_2, X_3, X_4, X_5)$.

Consider the simple Bayesian network in Fig. 2.2 and its *induced graphs*. An induced graph along an elimination order is obtained by moralizing the Bayesian network, arranging the nodes vertically according to the order, and from top to bottom recursively connecting each node's neighbors that appear later than itself. Dashed lines are induced arcs, and double arrows are commutable nodes. The width of a variable X along the order is the number of nodes succeeding X in the order and connected to X minus 1. The width of a graph is the maximum

width among all nodes, which is also called the induced width. Shaded nodes are those whose widths are maximal. An induced graph for the network's MPE problem is shown in part (b). We can solve the MPE problem of the network using an elimination order with induced width 1, which is also the treewidth. Part (c) shows the induced graph of a MAP problem. In the problem, we have to sum out X_3 and X_5 first, so the best elimination order has induced width 2. Notice that the network in this example is a simple polytree, for which belief updating and MPE are polynomial. However, because of the constrained width, MAP becomes an NP-hard problem. It is still possible to find valid orderings that interleave summation and maximization variables. However, Park [13] shows that there is always an elimination order with the same width in which all the maximizations are done last, and, hence, there is no benefit of interleaving summations and maximizations.

2.3 Previous Research

To solve the MAP problem for Bayesian networks, researchers have proposed various approaches, all of which are trying to sidestep its inherent complexity. The approach in [5] uses the *genetic algorithms* to approximate the best configuration of the MAP variables. Starting from an initial guess, the algorithm takes actions like crossover and mutation to explore the space of possible instantiations. It stops when a fixed number of iterations have been executed and then choose the best instantiation as the MAP solution. Dechter and Rish [6] propose a general scheme for probabilistic inference: *Mini-buckets*. A full mini-bucket algorithm is subject to the size of the largest potential created, which is equal to the constrained width of the MAP problem plus 1. Hence, the mini-bucket method sets a limit on the size of potentials. Whenever the size of a potential exceeds the limit, the mini-bucket method will create an approximate version of it instead. Park and Darwiche [14] propose an approach using *local search* to solve the MAP problem. The algorithm starts from an initial guess and then iteratively improves the solution by moving to a better neighbor. In a later paper [15], the authors improve the local search algorithm by means of *branch-and-bound depth-first systematic search* algorithm. The advantage of the improved algorithm is that it provides a guarantee on the optimality of the obtained solution. All of these algorithms could provide very efficient solutions for most of the MAP problems when the networks are not too large or complex. Another approximate algorithm proposed by Yuan et al. [21] is a Reheated Annealing MAP algorithm. It uses Markov Chain Monte Carlo methods to sample from the target distribution, and applies the reheated simulated annealing technique to simulate a nonhomogeneous Markov chain. It is somewhat slower on simple networks but it is able to handle very hard cases that the exact algorithm can not solve.

All of the above approaches alleviate to some degree the complexity of the

original problem. However, in face of large complex models, they often fail to provide good results, if any: the approach in [5] does not provide any guidance to explore the more probable spaces. The quality of the results of the mini-bucket method largely depend on the limit of the potential size. The algorithms in [14, 15] reduce the complexity of the MAP problems to treewidths, but they are still subject to the exponential search spaces introduced in the problems.

Later of my thesis, I will show the efficiency and accuracy of the DWA^* algorithm by comparing the simulation results of it with those of the local search, systematic search, and the AnnealedMAP .

Chapter 3

Asymmetries Among Joint Probability Distributions

A small fraction of states of a joint probability distribution can be expected to cover a large portion of the total probability space with the remaining states having practically negligible probability [7]. Theoretical discussion has been supplemented by simulation results. Let us give an concise introduction to the argument of the asymmetry among probability of the Joint Probability Distributions.

3.1 Preliminaries

3.1.1 Probabilistic Models

The essence of any probabilistic model is a specification of the joint probability distribution over the model's variables. i.e., probability distribution over all possible deterministic states of the model. It is sufficient for deriving all prior, conditional, and marginal probabilities of the model's individual variables.

Most modern textbooks on probability theory relate the joint probability distribution to the interactions among variables in a model by factorizing it, i.e., breaking it into a product of priors and conditionals. While theis view has its merits in formal expositions, it suggests viewing a probabilistic model as merely a numerical specification of a joint probability distribution that can be possibly algebraically decomposed into factors. This clashes with our intuition that whatever probability distribution we observe, they are a product of structural, causal properties of the domain. Causal interactions among variables in a system determine the observed probabilistic dependence and, in effect, the joint probability distribution over all model's variables. An alternative view of a joint probability distribution is, therefore, that it is composable from rather

than decomposable into prior and conditional probability distribution. In this view, each of these distributions corresponds to a causal mechanism acting in the system. This reflects the process of constructing joint probability distributions over domain models in most practical situations.

Since insight obtained from two modeling tools: Bayesian belief networks (BBNs) (Pearl, 1988) and probability trees may prove useful for the reader, I will show how they both represent a simple uncertain model involving a common activity of a clinician interpreting the result of a screening test for a disease. This model contains two binary variables: disease and test. The outcomes of variable disease, d and $\overline{d}$, stand for disease present and disease absent respectively. The outcomes of variable test, t and $\overline{t}$, stand for test positive and test negative respectively A BBN representing this problem, shown in Figure 3.1, reflects the qualitative structure of the domain, showing explicitly dependences among variables Each variable is characterized by a probability distribution conditional on its predecessors or by a prior probability distribution if the variable has no predecessors. Figure 3.1 shows also a probability tree encoding the same problem. Each node in this tree represents a random variable and each branch originating from that node a possible outcome of that variable. Each complete path starting at the root of the tree and ending at a leaf corresponds to one of the four possible deterministic states of the model.

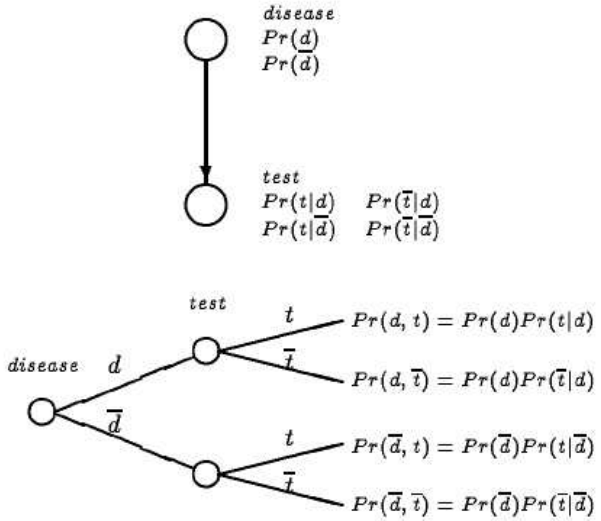

Figure 3.1: Two probabilistic representations of the screening test problem Bayesian belief network (upper) and probability tree (lower).

The probabilities of various states of a model can be easily retrieved in BBNs and probability trees by multiplying out the prior and conditional probabilities of individual variables In the models of Figure 3.1, we multiply the priors of various outcomes of disease by the conditionals of respective outcome of test given presence or absence of disease.

3.1.2 State Probabilities

First we choose at random one state of a model that consists of n variables $X_1, X_2, X_3, \ldots, X_{n-1}, X_n$. We choose this state equationally from among all possible states, regardless of its probability. As a state is an assignment of each of the model's n variables, one way of looking at this selection process is that we are traversing the probability tree representing the model from its root to one of its leaves taking at each step one of the possible branches with equal probability. This amounts to a random choice of one outcome from among the outcomes of each of the variables. The probability of p of a selected state is equal to the product of conditionals of each of the randomly selected outcomes. It is equal for our selected state to $p = Pr(d, t) = Pr(d)Pr(t \mid d)$. Generally, if we denote p_i to be the conditional(or prior) probability of the randomly selected outcome of variable X_i, We have

$$p = p_1 p_2 p_3 \ldots p_{n-1} p_n = \prod_{i=1}^{n} p_i \ . \tag{3.1}$$

In random selection of state, We chose each p_i to be one number from among the probabilities of various outcomes of variable X_i. We can, therefore, regard each p_i as a random variable taking equiprobable values from among the probabilities of the outcomes of variable X_i.

Obviously, the distribution of p_i is not always independent from the distribution of p_j, when $i \neq j$, as the outcomes of some variables may impact the conditional probability distributions of other variables. Selection of p_i within its distribution is nevertheless independent of any other p_j, when $i \neq j$. Specifically, in Bayesian networks, which are depicted by conditional probability distribution between any pair of nodes connected by arcs, the state of each inner node depends on the outcomes of its causal ancestors. The exact form of this distribution is a property of the mechanism and is independent on anything else in the systems.

By taking the logarithm of both side of equation 3.1 I can obtain that:

$$Lnp = ln \prod_{i=1}^{n} p_i = \sum_{i=1}^{n} lnp_i \ . \tag{3.2}$$

As for each i, p_i is a random variable, its algorithm lnp_i is also a random variable for p_i is a random variable. The asymptotic behavior of a sum of random

variables is relatively well understood and addressed by a class of limit theorems known collectively as Central Limit Theorem. When the number of components of the sum approaches infinity, the distribution of the sum approaches normal distribution, regardless of the probability distribution of the individual components. Even though in any practical case we will be dealing with a finite number of variables, the theorem gives a good approximation even the number of variables is small.

3.1.3 Central Limit Theorem

Central limit theorem (CLT) is one of the fundamental and most robust theorems of statistics, applicable to a wide range of distributions. It was originally proposed for Bernoulli variables, then generalized to independent identically distributed variables, then to non-identically distributed, and to some cases where independence is violated. Extending the boundaries of distributions to which CLT is applicable is one of active areas of research in statistics. CLT is so robust and surprising that it is sometimes referred to as "order out of chaos" (de Finetti, 1974)

One of the most general forms of CLT is due to Liapounov (to be found in most statistics textbooks)

Theorem 1 *Let $X_1, X_2, X_3, \ldots, X_n$ be a sequence of n independent random variables such that $E(X_i) = \mu_i$, $E((X_i - \mu_i)^2) = \sigma_i^2$, and $E(\mid X_i - \mu_i \mid^3) = \omega_i^3$ all exist for every i. Then their sum, $Y = \sum_{i=1}^{n} X_i$ is asymptotically distributed as $N(\sum_{i=1}^{n} \mu_i, \sum_{1}^{n} \sigma_{i=1}^2)$, provided that*

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} \omega_i^3}{(\sum_{i=1}^{n} \sigma_i^2)^{3/2}} = 0. \tag{3.3}$$

This condition is satisfied for any distribution for which μ and σ exist and the theorem reduces to Lindeberg and Levy's version of CLT (also reported in most textbooks).

Returning to Equation 3.2, we have by the CLT, that assuming that the preconditions of CLT are satisfied, the sum on the right side is in the limit normally distributed. If ln$p$ is normally distributed, then p itself must be drawn from a lognormal distribution.

3.2 Properties of the Joint Probability Distribution

CLT captures the growth of a process showing strong regularity and satisfying certain independence conditions, and these conditions are reasonably satisfied in the process of constructing a joint probability distribution [7]. In what follows, I will be showing the properties of the logarithm of the distribution.

Let a model consist of n variables $X_1, X_2, X_3, \ldots, X_n$, having $k_1, k_2, k_3, \ldots, k_n$, states respectively $(1 \leq i \leq n)$. For any single state, we can apply the Central Limit Theorem to equation 3.2, viewing each p_i as an independent random variable. The value of p_i will be the probability of a randomly selected outcome of variable X_i. Let the mean and the variance of the distribution of p_i be μ_i and σ_i^2 respectively. The logarithm of p, the probability of an individual state, obtained by multiplying priors and conditionals of individual variables in then distributed as $lnp \sim N(\sum_1^n \mu_i, \sum_1^n \sigma_i^2)$.

The density function $f(lnp)$ is:

$$f(lnp) = \frac{1}{\sqrt{2\pi \sum_{i=1}^n \sigma_i^2}} exp \frac{-(lnp - \sum_{i=1}^n \mu_i)^2}{2 \sum_{i=1}^n \sigma_i^2}; . \tag{3.4}$$

Example. In order to give readers a direct impression, I cited one of the most classical simulation results that has been reported [Druzdzel 1994, Conference of Uncertainty in Artificial Intelligence].

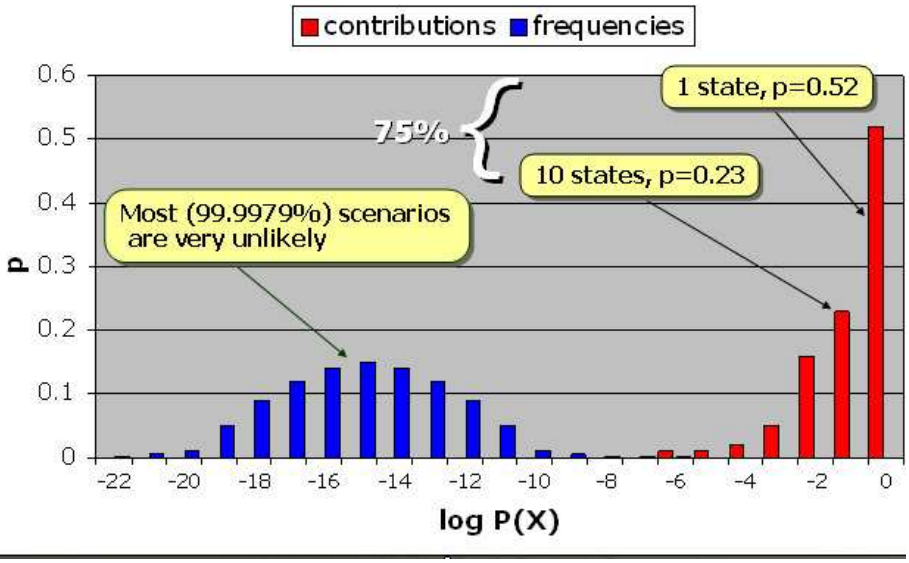

Figure 3.2: Plot of the probability density function $f(lnp)$(in blue) and contributions of each scenario's probability $pf(lnp)$(in red).

In the Bayesian network ALARM [4], the researcher chose a subset of 13 variables as MAP variables, which led to 525,312 scenarios.

The sum of the probability of each scenario is **1**, and we can see that among all 525,312 scenarios, there was only one scenario with the probability of 0.52,

and other ten scenarios with a sum of the probability to 0.23. From the perspective of MAP Problem, there are 99.9979% scenarios with practically negligible probability which are very unlikely to be the solution.

3.3 Tentative Research Related to Joint Probability Distributions

The estimation of joint probability distributions of MAP variables given evidences was also a hot point of research. Tomasz Loboda, my colleague in the Decision Systems Laboratory (DSL) reported his research result that the estimation of the *mean* of the lognormal distribution is the **exact** value, while the error of the estimation of *variance* was up to 67% in terms of percentage error.

During the first three months's research in DSL, the concentration of my research was proposed to exploit the estimation of joint probability distributions of MAP variables given evidences.

It was a branch-and-bound search algorithm combined with the estimation of joint probability distribution. The basic idea was in each search path of the probability tree, I could estimate the upper bound of the scenario by using the integration of function 3.4. If the upper bound is smaller than the probability of the best scenario found so far, then the current search path should be cut.

The requirement to the accuracy of the upper bound was so strict that the estimation errors led the algorithm to be a quite frustrating one, however, the experiences accumulated from developing the branch-and-bound algorithm greatly helped me to find new inspiration for the Dynamic Weighting A^* Search.

Chapter 4

Solving MAP using Dynamic Weighting A^* Search

I present in this section an algorithm for solving MAP using Dynamic Weighting A^* search, which incorporates the *dynamic weighting* [16] in the heuristic function, *relevance reasoning* [8] and *dynamic ordering* in the search tree.

The remainder of this chapter is organized as follows. First, I introduce the A^* Search. Second, I compare the A^* search algorithm with the branch-and-bound algorithm which is also suitable for solving the MAP problem, and show why it is superior. Third, I introduce our method of composing the heuristic function $h(n)$. Fourth, I analyze the efficiency and accuracy of the dynamic weighting A^* search algorithm and the situation when over-estimate happened. Finally, I discuss two techniques for improving the efficiency of the algorithm.

4.1 A^* search

The A^* search is **complete, optimal**, and optimally efficient among all search algorithms is rather satisfying [18].

The MAP problems can be solved by A^* search in the probability tree that is composed of all the variables in the MAP set. The nodes in the search tree represent partial assignments of the MAP variables $\mathbf{M}$. The root node represents an empty assignment. Each MAP variable will be instantiated in a certain order. If a variable $\mathbf{x}$ in the set of the set of MAP variables $\mathbf{M}$ is intantiated at the i*th* place using its j*th* state, it will be denoted as $\mathbf{M}_{ij}$. Leaves of the search tree correspond to the last MAP variable that has been instantiated. The vector of instantiated states of each MAP variable is called an *assignments* or a *scenario*.

I compute the probability of assignments while searching the whole probability tree using chain rule. For each inner node, the newly instantiated node will be added into the evidence set, i.e., the evidence set will be extended to $\mathbf{M}_{ij} \cup \mathbf{E}$.

Then the probability of the MAP problem which consists of **n** MAP variables can be presented as follows:

$$\begin{aligned} P(\mathrm{M} \mid E) &= P(M_{ni} \mid M_{1j}, M_{2k}, \ldots M_{(n-1)t}, E) \\ &\quad \ldots P(M_{2k} \mid M_{1j}, E) P(M_{1j} \mid E) \ . \end{aligned}$$

Each posteriori probability at the righthand side of the equation above can be computed by the jointree algorithm [12] efficiently. The jointree algorithm is a very complex while efficient algorithm for computing the posteriori probability of MAP variables given evidences in Bayesian networks which has been developed and included in the SMILE library. If the readers are interested in the jointree algorithm, I refer the reader to the article of Lauritzen and Spiegelhalter [12] for more information.

Suppose we are in the x*th* layer of the search tree and preparing for instantiating the x *th* MAP variables. Then the function above can be rewritten as follows:

$$\begin{aligned} P(\mathrm{M} \mid E) = \overbrace{P(M_{ni} \mid M_{1j} \ldots M_{(n-1)t}, E) \ldots P(M_{(x+1)z} \mid M_{xy} \ldots E)}^{b} \\ \cdot \underbrace{P(M_{xy} \mid M_{1j}, M_{2k} \ldots M_{(x-1)q}, E) \ldots P(M_{1j} \mid E)}_{a} \end{aligned}$$

(4.1)

The general idea of the Dynamic Weighting A^* Search algorithm is that during the search, in each inner node of the probability tree, I can compute the value of item (a) in the above function *exactly*. I can estimate the heuristic value of the item (b) for the MAP variables that have not been instantiated given the initial evidence set and the MAP variables that have been intantiated as the new evidence. In order to fit the typical format of the cost function of A^* Search, I just take the logarithm of the equation above, which will not change its monotonicity. Then I can get $f(n) = g(n) + h(n)$, where $g(n)$ and $h(n)$ are obtained from the logarithmic transformation of items (a) and (b) respectively. $g(n)$ gives the exact cost from the start node to node in the n*th* layer of the search tree, and h(n) is the estimated cost of the best search path from the n*th* layer to the leaf nodes of the search tree. In order to guarantee the optimality of the solution, $h(n)$ should be *admissible*, which in this case means that it should be an upper-bound on the value of any assignment with the currently intantiated MAP variables as its elements.

4.2 Efficiency Comparison between A^* and Branch-Bound Search for MAP Problems

A^* Search is closely related to the *branch-and-bound* techniques. For the MAP problems, the efficiency of the search algorithm is dominated by the number of nodes in the probability tree that are instantiated. In order to compare the efficiency of the two search algorithms for the MAP problems, I will first introduce some definitions.

Definition 5 *: An algorithm A_1 is said to* dominate *an algorithm A_2 if every node expanded by A_1 is also expanded by A_2. I will also use the phrase* more efficient than *interchangeably with* dominates *[16].*

Let s_{max} be the most probable assignment and its probability be P_{max}. Let $P_{BestSoFar}$ denote the probability of the best assignment that we have found so far, which is less or equal to the P_{max}.

Theorem 2 *Given the same cost function f(x), A^* dominates branch and bound on MAP problems.*

Proof: The condition for cutting the current search path in "branch and bound" search is $f(x) < P_{BestSoFar}$. So the whole search space can be denoted by:

$$\begin{aligned} S_1 \quad = \quad & \{x : f(x) \ \geq \ P_{BestSoFar} \\ & \cup \ x \in \textit{optimal assignment}\} \ . \end{aligned}$$

For the A^*, consider a node y that is currently in the search frontier which is also an element of the vector of the optimal scenario. With the admissibility of the f(y), which guarantees that f(y) an upper-bound on the probability of the optimal solution, we have $f(y) \geq P_{max}$. For all nodes x that on the search path other than the optimal assignment, we have $f(x) \geq f(y)$. The whole search space can be denoted by:

$$\begin{aligned} S_2 \quad = \quad & \{x : f(x) \ \geq \ f(y) \\ & \cup \ x \ \in \ \textit{optimal assignment}\} \end{aligned}$$

Given that $P_{BestSoFar} \leq P_{max}$ and $f(y) \geq P_{max}$, we have $f(y) \geq P_{BestSoFar}$, which implies $S_2 \subseteq S_1$, i.e., that the A^* search dominates the branch-and-bound search. □

4.3 Heuristic Function with Dynamic Weighting

The A^* Search is known for its completeness and optimality. For each search step, I only expand the node in the frontier with the largest value of $f(n)$.

Definition 6 *A heuristic function h_2 is said to be* more informed than h_1 *if both are admissible and h_2 is closer to the optimal cost. For the MAP problem, the probability of the optimal assignment $P_{opt} < h_2 < h_1$.*

Theorem 3 *If h_2 is more informed than h_1 then A_2^** dominates *A_1^* (Nilsson). [16]*

The **power** of the heuristic function is measured by the amount of pruning induced by $h(n)$ and depends on the accuracy of this estimate. If $h(n)$ estimates the completion cost precisely ($h(n) = P_{opt}$), then A^* will only expand nodes on the optimal path. On the other hand, if no heuristic at all is used, (for the MAP problem this amounts to $h(n) = 1$), then a uniform-cost search ensues, which is far less efficient. So it is critical for us to find an *admissble* and *tight* $h(n)$ to get both accurate and efficient solutions for MAP.

Greedy Guess

If each variable in the MAP set **M** is conditionally independent of all the rest of MAP variables (this is called *exhaustive independence*), then the MAP problem amounts to a simple computation based on the *greedy* chain rule. I instantiate the MAP variable in the current search layer to the state with the largest probability and repeat this for each of the remaining MAP variables one by one. The probability of MAP is then

$$P(M|E) = \prod_{i=1}^{n} \max_j P(M_{ij}|M_{(i-1)k} \ldots M_{1m}, E) \ . \tag{4.2}$$

The requirement of exhaustive independence is too strict for most of the MAP problem to be calculated by using the function above. Simulation results show that in practice, when this requirement is violated, the product is still extremely close to the MAP probability [21]. This suggests using it as an ϵ-admissible heuristic function [16].

The curve *Greedy Guess Estimate* in Figure 4.1 shows that with the increase of the MAP variables, the ratio between the greedy guess and the accurate estimate of the optimal probability diverges from the ideal ratio **1.0** although not always monotonically.

Dynamic Weighting

Since the greedy guess is a tight lower bound on the optimal probability of MAP, it is possible to compensate for the error between the greedy guess and the optimal probability. I can do this by adding a weight to the greedy guess such that the product of them is equal or larger than the optimal probability for each inner node in the search tree. This yields an *ϵ-admissible* heuristic

function that I need in order to find the optimal solutions. This assumption can be represented as follows:

$$\exists\epsilon\{\forall P_{GreedyGuess} * (1+\epsilon) \geq P_{opt} \wedge \forall P_{GreedyGuess} * (1+\epsilon^{'}) \geq P_{opt} \Rightarrow \epsilon < \epsilon^{'}\}$$

where ϵ is the minimum weight that can guarantee the heuristic function to be admissible. Figure 4.1 shows that if I just keep ϵ constant, neglecting the changes of the estimate accuracy with the increase of the MAP variables, the estimate function and the optimal probability can be represented by the curve *Constant Weighting Heuristic*. Obviously, the problem with this idea is that it is less informed when the search progresses, as there are fewer MAP variables to estimate.

Dynamic Weighting (Pohl, 1973) is an efficient tool for improving the efficiency of A^* Search. If applied properly, it will keep the heuristic function admissible while remaining tight on the optimal probability. For MAP, in the shallow layer of the search tree, we get more MAP variables than the deeper layer for estimate. Hence the greedy estimate will be more likely to diverge from the optimal probability. I propose the following Dynamic Weighting Heuristic Function for the x*th* layer of the Search tree of n MAP variables:

$$\begin{aligned} h(x) &= P_{GreedyGuess} \cdot (1 + \alpha\frac{n-(x+1)}{n}) \\ & \quad (\alpha \geq \epsilon)\ . \end{aligned}$$

Rather than keeping the weight constant throughout the search, I dynamically change it so as to make it less heavy as the search goes deeper. In the last step of the search ($x = n-1$), the weight will be zero, since the Greedy Guess for only one MAP variable is exact and then the cost function f(n-1) is equal to the probability of the assignment. Figure 4.1 shows an empirical comparison of greedy guess, constant, and dynamic weighting heuristics against accurate estimate of the probability. We see that the dynamic weighting heuristic becomes more informed than constant weighting. In our experiments, I set α to be 1.0, which is tested to be a quite conservative while efficient parameter.

4.4 Searching with Nonadmissible Heuristics for MAP Problem

Let us have a closer look at the conditions under which the algorithm fails to achieve optimality. Suppose there are two candidate assignments: s_1 and s_2 with probability p_1 and p_2 respectively, among which s_2 is the optimal assignment that the algorithm fails to find. And s_1 is now in the last step of search which will lead to a suboptimal solution. I skip the logarithm in the function for the sake of clarity here (then the cost function f is a product of transformed g and h instead of their sum).

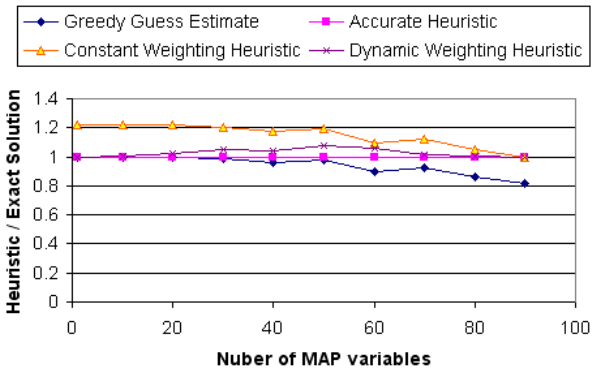

Figure 4.1: Constant Weighting Heuristic and Dynamic Weighting Heuristic based on Greedy Guess.

$$f_1 = g_1 \cdot h_1 \text{ and } f_2 = g_2 \cdot h_2$$

The error introduced by a non-admissible h_2 is $f_1 > f_2$. The algorithm will then find s_1 instead of s_2, i.e.,

$$f_1 > f_2 \Rightarrow g_1 \cdot h_1 > g_2 \cdot h_2.$$

Since s_1 is now in the last step of search, $f_1 = p_1$ (Section 3.3.2). Now suppose that I have an *ideal* heuristic function $h_2^{'}$, which leads to $p_2 = g_2 \cdot h_2^{'}$. Then I have:

$$\frac{g_1 \cdot h_1}{p_2} > \frac{g_2 \cdot h_2}{g_2 \cdot h_2^{'}} \Rightarrow \frac{p_1}{p_2} > \frac{g_2 \cdot h_2}{g_2 \cdot h_2^{'}} \Rightarrow \frac{p_1}{p_2} > \frac{h_2}{h_2^{'}}$$

It is clear that only when the ratio between the probability of suboptimal assignment and the optimal one is larger than the ratio between the nonadmissible heuristic function and the ideal one, may the algorithm find a suboptimal solution.

Because of large asymmetries among probabilities that are further amplified by their multiplicative combination [7], I can expect that for most of cases, the ratios between p_1 and p_2 are far less than **1**. Even though the heuristic function will sometimes break the rule of admissibility, if only the greedy guess is not too divergent from the ideal estimate, the algorithm will still not diverge from the optimal probability. Our simulation results also proved the robustness of the algorithm in finding optimal solutions.

4.5 Improvements to the Algorithm

There are two main techniques that I used to improve the efficiency of the basic A^* algorithm.

4.5.1 Relevance Reasoning

The main problem faced by the decision-theoretic approach is the complexity of probabilistic reasoning. The critical factor in exact inference schemes for Bayesian networks is the topology of the underlying graph and, more specifically, its connectivity. The framework of relevance reasoning ([8] is an accessible summary of the relevant techniques) is based on d-separation and other simple and computational efficient techniques for pruning irrelevant parts of a Bayesian networks and can yield sub-networks that are smaller and less densely connected than the original network. Relevance reasoning is an integral part of the SMILE library on which the implementation of our algorithm is based.

For MAP, our focus is the set of variables **M** and the evidence set **E**. Parts of the model that are probabilistically independent from the nodes in **M** given the observed evidence **E** are computationally irrelevant to reasoning about the MAP problem.

4.5.2 Dynamic Ordering

As the search tree is constructed dynamically, I have the freedom to order the variables in a way that will improve the efficiency of the DWA^* search. Expanding nodes with the largest asymmetries in marginal probability distribution leads to early cut-off of less promising branches of the search tree. I use the entropy of the marginal probability distributions as a measure of asymmetry.

The basic concept of entropy in information theory has to do with how much randomness is in a signal or in a random event and how much information is carried by the signal. An alternative way to look at this is to talk about how asymmetric the probability of different states of a MAP variable is.

Claude E. Shannon defines entropy in terms of a discrete random event x, with possible states 1..n as:

$$H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$$

Theoretically, the lower the entropy of a probability distribution is, the more asymmetric its probability of different states will be.

The first step is to compute entropy **H** for each variable in **M** that has not been instantiated. This can be computed from the marginal potential of each variable in **M** efficiently. I then choose the variable with the least entropy as the next variable to be instantiated.

The reason that I take this way is not difficult to understand. Let us look back at the equation (4.1). When deciding the instantiation order that leads to the "*early-cut*", I always select the variable that could *diverge* the value of item (a) after instantiation. The result of this is that the probability of different branches in the search tree will be quickly driven to be two polars, a small portion of them with very large values of item (a) and most of the others with very small values, which are named *less-promising branches* and more likely to generate scenarios with very little probabilities.

Hence, by dynamic ordering, the DWA^* algorithm works even better because of typically enormous asymmetries among the probabilities of individual scenarios will be found even in shallow layer of the search tree. Then the DWA^* algorithm will not spend a large amount of time discriminating among paths whose costs do not vary significantly from each other, which leads to great reduction of the search space.

4.6 DWA^* Algorithm

Given the above discussion, I outline the DWA^* Algorithm in Fig. 4.2. When one of the elements of the search frontier reaches the leaf node of the search tree the DWA^* algorithm will terminate, and I can take the final configuration as the output.

Algorithm: DWA^*
Input: Bayesian network B, a set of MAP variables M, a set of evidence variables E, Weight α;
Output: The most probable configuration of M.

1. Call Greedy Guess at search layer 0, and set L=GreedyGuess(0).
2. **while** current search layer is not the bottom of the search tree:
3. Select the node M_l with the largest value of the cost function $F(x)$ in the search frontier.
4. Set all the instantiated variables before M_l to be new Evidences nodes.
5. Set all uninstantiated variables in M to be Target Nodes.
6. Update beliefs of B.
7. Sort all uninstantiated variables in M in a increasing order of Entropy, among which M_s has the least Entropy.
8. Expand the node M_l by using M_s as its child nodes.
9. Insert new generated child nodes into the search frontier.
10. Clear Evidences.
11. Unset Target Nodes.

    end while
12. Output the best configuration found and its probability.

Figure 4.2: The DWA^* algorithm.

Chapter 5

Implementation

The total programming work of implementing the DWA^* algorithm was accomplished by myself using C++ in the Windows programming environment.

The remainder of this chapter is organized as follows. First, I introduce the Structural Modeling, Inference, and Learning Engine (SMILE) library by which the DWA^* algorithm is supported. Second, I introduce my programming work on the DWA^* algorithm which will be included in the new version of SMILE for solving the MAP problem. Finally, I will give a brief introduction on the approaches that I used to compare the DWA^* algorithm with the other existent algorithms.

5.1 SMILE

Structural Modeling, Inference, and Learning Engine (SMILE) is a fully platform independent library of C++ classes implementing graphical probabilistic and decision-theoretic models, such as Bayesian networks, influence diagrams, and structural equation models. Its individual classes, defined in SMILE Applications Programmer Interface, allow to create, edit, save, and load graphical models, and use them for probabilistic reasoning and decision making under uncertainty. These classes are accessible from C++ or (as functions) from C programming languages. As most implementations of programming languages define a C interface, this makes SMILE accessible from practically any language on any system. Also SMILE may be embedded in programs that use graphical probabilistic models as their reasoning engines. Furthermore, models developed in SMILE can be equipped with a user interface that suits the user of the resulting application most. Additional to the SMILE platform is the development of SmileX, an ActiveX Windows component that allows SMILE to be accessed from any Windows programming environment, including World Wide Web pages.

5.2 DWA^* Class

The DWA^* algorithm was implemented in the new Class named "DWAstar". The introduction to this class will be given in terms of two subsections: Data member and Member function.

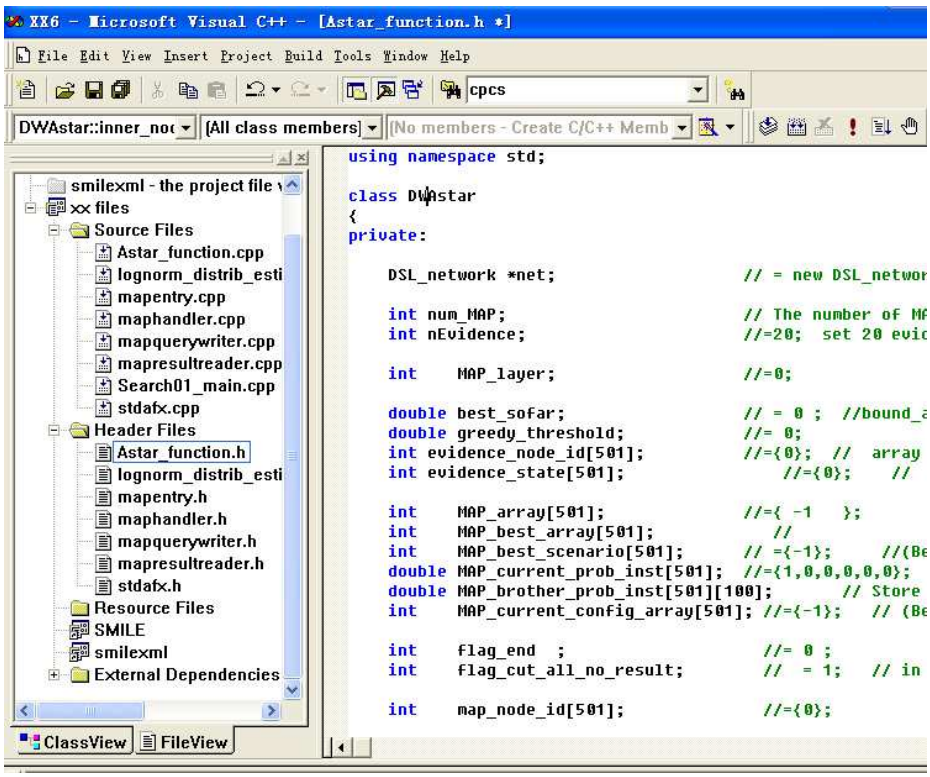

Figure 5.1: DWAstar Class.

5.2.1 Data member

In Figure 5.2, I listed main data member of the Class "DWAstar":

- **DSL_network * net**
  The Bayesian networks in which the MAP problem is.
- **num_MAP**
  The number of variables in MAP set.

- **nEvidences**
  The number of variables in Evidences set.
- **MAP_layer**
  The current search layer in the probability tree.
- **evidence_node_id[501]**
  The handle of each Evidence variable.
- **evidence_state[501]**
  The state of each Evidence variable.
- **MAP_array[501]**
  The handle of each MAP variable.
- **MAP_best_array[501]**
  The handle of each MAP variable in the final scenario that the algorithm returns. Since the sequence of MAP variables will be dynamically ordered, the sequence of MAP variables in MAP_best_array[501] is generally different from that in MAP_array[501].
- **MAP_current_config_array[501]**
  The handle of each MAP variable in the current search path.
- **flag_end**
  The variable to testify whether the algorithm should be terminated.
- **map_node_id[501]**
  The handle of the MAP variables that have been randomly generated for testing the algorithm.
- **largest_product_index**
  The index of the search path with the largest value of cost function in the search frontier.
- **Final_Joint_Probability**
  The probability of the scenario that the algorithm returns.
- **greedy_threshold**
  The probability that the greedy guess generated before the search. It is a tight lower bound on the MAP problem.
- **inner_node**
  The struct that records each node in the search tree.
- **inne_points**
  The vector that records the search frontier.

5.2.2 Member function

Here listed in Figure 5.3 are the main member functions of the Class "DWAstar", among which I will show Greedy_guess, Estimate_inner_node, and A_Star_Search in detail which are key components of the DWA^* algorithm:

- **DWAstar (DSL_network * theNet, intnumber_Evidence, intnumber_MAP)**
  Constructor of the Class.
- **void Set_evidence(int a)**
  Randomly generate **a** evidence variable, and randomly select one of all its possible states as its state.
- **void Set_map(int b)**
  Randomly generate **b** MAP variable.
- **double Entropy(int node_id)**
  Compute and return the Entropy of the MAP variable.
- **void Order_single_MAP_variables_entropy(int MAP_layer)**
  Order the MAP variables in terms of Entropy of them, and shift the variable with the least Entropy to the first position to be instantiated.
- **double Get_greedy_threshold()**
  Return the lower bound on the MAP problem.
- **void Show_MAP_array()**
  Print out the handle of each MAP variable.
- **void Set_One_Evi_Node(int Evi_id, int Evi_state, int Squence)**
  Set only one Evidence node.
- **void Set_One_MAP_Node(int MAP_id, int Squence)**
  Set only one MAP node.
- **void Show_all_initial_variables()**
  Show all the handles of MAP variables and Evidence variables.
- **void Show_all_evidence_and_state()**
  Show all evidence variables and their states correspondingly.
- **void Change_num_of_Evidence(int num_evidence)**
  Assign the value of the number of evidence variables.
- **void Change_num_of_MAP(int num_map)**
  Assign the value of the number of MAP variables.
- **double Compute_PRE(int num_of_evidences)**
  Compute the joint probability of all evidence variables.

- **void Set_greedy_threashold(double a)**
  Set the value of the *greedy_threashold*, it is a lower bound on the MAP problem.

- **void Show_greedy_threashold()**
  Print out the value of the *greedy_threashold*.

- **void Extend_node(int MAP_layer)**
  Expand the current search path to the next layer in the probability tree.

- **double Greedy_guess(int MAP_layer)**
  Return the posteriori probability of the MAP variables that have not been instantiated, given the evidences and the MAP variables that have been instantiated. Readers can resort to the codes shown in figure 5.4 for detail.

- **double Estimate_inner_node(int MAP_layer)**
  Return the value of the heuristic function $h(x = MAP\_layer)$. The computation is fulfilled by using Dynamic Weighting Techniques, which is the critical part of the DWA^* search algorithm. Readers can resort to the codes shown in figure 5.5 for detail.

- **double A_Star_Search(int num_of_MAP)**
  Search the probability tree. Readers can resort to the codes shown in figure 5.6 for detail.

```
class DWAstar
{private:
    DSL_network *net;
    int      num_MAP;
    int      nEvidence;
    int      MAP_layer;
    int      evidence_node_id[501];
    int      evidence_state[501];
    int      MAP_array[501];
    int      MAP_best_array[501];
    int      MAP_best_scenario[501];
    int      MAP_current_config_array[501];
    int      flag_end    ;
    int      map_node_id[501];
    int      Largest_product_index;
    int      flag_frontier_only_one_or_none;
    double   MAP_current_prob_inst[501];
    double Final_Joint_Probability;
    double greedy_threshold;
    struct inner_node { int cut_layer;
                        double cut_probability;
                        double estimate_value;
                        int cut_map_node_full[501];
                        int cut_map_current_state[501];
    }   ;
    vector<inner_node>   inner_points ;
```

Figure 5.2: Data member of the DWAstar Class.

```
public:
    Astar(DSL_network *theNet, int number_Evidence, int number_MAP);     //Declare
    void Set_evidence(int a)                                             ;  //Declare
    void Set_map(int b)                                                  ;
    void Extend_node (int & MAP_layer )                                  ;  //Declare
    double Entropy(int node_id)                                          ;  //Declare
    void Order_single_MAP_variables_entropy( int MAP_layer) ;            //Declare
    double Estimate_inner_node(int MAP_layer)                            ;  //Declare
    double Greedy_guess(int MAP_layer )                                  ;  //Declare
    double Get_greedy_threshold()                                        ;  //Declare
    void Show_MAP_array()                                                ;  //Declare
    double A_Star_Search(int num_of_MAP)                                 ;  //Declare
    void Set_One_Evi_Node(int Evi_id, int Evi_state, int Squence); //Declare
    void Set_One_MAP_Node(int MAP_id, int Squence)                       ;  //Declare
    void Show_all_initial_variables()                                    ;  //Declare
    void Show_all_evidence_and_state()                                   ;  //Declare
    void Change_num_of_Evidence(int num_evidence)                        ;  //Declare
    void Change_num_of_MAP(int num_map)                                  ;  //Declare
    double Compute_PRE( int num_of_evidences)                            ;  //Declare
    void Set_greedy_threashold(double a)                                 ;  //Declare
    void Astar::Show_greedy_threashold()                                 ;  //Declare
};
```

Figure 5.3: Member function of the DWAstar Class.

```
double DWAstar::Greedy_guess(int MAP_layer )
{       int i, index_guess;
        int guess_current_config_array[500];
        double temp_prob_layer = 1;
        double guess_brother_prob_inst[100];
        net->SetTarget(MAP_array[MAP_layer+1]);
        if(MAP_layer == 0)
        net->UpdateBeliefs();
        if(MAP_layer>0)
        {       DSL_node * mNode1 = net->GetNode(MAP_array[MAP_layer]);
                mNode1->Value()->SetEvidence(MAP_current_config_array[MAP_layer])
                net->UpdateBeliefs();
                Update_Beliefs;  }
        for( i=MAP_layer+1 ; i<=num_MAP; i++)           // i used to be 1
        {       DSL_Dmatrix *theMatrix3;
                theMatrix3 = net->GetNode(MAP_array[i])->Value()->GetMatrix();
        for(int j=0;j<net->GetNode(MAP_array[i])->Definition()->GetNumberOfOutcomes();j++)
                 guess_brother_prob_inst[j] = (theMatrix3->Subscript(j) );
                 double tem = guess_brother_prob_inst[0];
                 index_guess=0;
         for( j=1;j<net->GetNode(MAP_array[i])->Definition()->GetNumberOfOutcomes();j++)
         {       if (guess_brother_prob_inst[j]>tem)
                  {tem = guess_brother_prob_inst[j];
                   index_guess=j; }    }
                   guess_current_config_array[i] = index_guess;
                   assert(guess_brother_prob_inst[index_guess]>=0.0
   temp_prob_layer = temp_prob_layer *    guess_brother_prob_inst[index_guess];
   if (MAP_layer>0)
       {       double coe;
               double alafa = 10.0;
                    coe=      temp_prob_layer * (1 + alafa * (num_MAP - i ) / num_MAP     ) ;
               if (coe * MAP_current_prob_inst[MAP_layer] < greedy_threshold && i != num_MAP)
                           return(0);
       }// end if (MAP_layer>0)
                  net->UnSetTarget(MAP_array[i]);
                 DSL_node * mNode = net->GetNode(MAP_array[i]);
               mNode->Value()->SetEvidence(guess_current_config_array[i]);
       if( (i+1)<num_MAP)
       {net->SetTarget(MAP_array[i+1]);
        net->UpdateBeliefs();     }
       } // end for
       for( i = MAP_layer+1; i<=num_MAP; i++)

         { DSL_node *mNode1 = net->GetNode(MAP_array[i]);
             mNode1->Value()->ClearEvidence();
           net->SetTarget(MAP_array[i]); }
             net->UpdateBeliefs(); }
return (temp_prob_layer);
}
```


Figure 5.4: *double Greedy_guess(int MAP_layer)*.

```
double DWAstar::Estimate_inner_node(int MAP_layer)
{
    double estimate_value = 0;
    double greedy_value=0;
    double alafa = 1.0 ;
    greedy_value = Greedy_guess( MAP_layer );
    estimate_value = greedy_value * (1 + alafa * (num_MAP - (MAP_layer +1 )) / num_MAP    ) ;
    return (estimate_value);
}
```

Figure 5.5: *double Estimate_inner_node(int MAP_layer)*.

```
double DWAstar::DWA_Star_Search(int num_of_MAP)
{    time_t start,finish;
      num_MAP = num_of_MAP;
      flag_frontier_only_one_or_none = 1;
     for(int i2=1 ; i2<=num_MAP; i2++)
      net->SetTarget(MAP_array[i2]);            //1 set target

net->UpdateBeliefs();
Order_single_MAP_variables_entropy(MAP_layer) ;
if(net->GetNode(MAP_array[MAP_layer+1])->Definition()->GetNumberOfOutcomes()>60    &&  MAP_layer
< (num_MAP-1) )
{ Order_single_MAP_variables_entropy(MAP_layer+1) ;
      int temp_element
     temp_element = MAP_array[MAP_layer+1];
      MAP_array[MAP_layer+1] = MAP_array[MAP_layer+2];
      MAP_array[MAP_layer+2] = temp_element;}

Extend_node (MAP_layer);
while (MAP_layer != num_MAP)
      { Largest_product_index = 0;
if(inner_points.size() > 1)
flag_frontier_only_one_or_none = 0;
for(int i =1; i<inner_points.size() ;i++      )
{if(inner_points[i].cut_probability                    *                    inner_points[i].estimate_value                    >
inner_points[Largest_product_index].cut_probability * inner_points[Largest_product_index].estimate_value )
Largest_product_index = i;}
MAP_layer = inner_points[Largest_product_index].cut_layer ;// (1)
MAP_current_prob_inst[MAP_layer] = inner_points[Largest_product_index].cut_probability;//(2)
      for(int   t=1;t<= num_MAP; t++)
      {MAP_array[t] = inner_points[Largest_product_index].cut_map_node_full[t] ;   //(4)
       MAP_current_config_array[t] = inner_points[Largest_product_index].cut_map_current_state[t]; //(5)
      }
//3 set_evidence here
      for(int j=1;j<=MAP_layer;j++)
      { net->UnSetTarget(MAP_array[j]);     // unset_target
       DSL_node * mNode = net->GetNode(MAP_array[j]);
       mNode->Value()->SetEvidence(MAP_current_config_array[j]);            }
       for( i=MAP_layer+1 ; i<=num_MAP; i++)      //from (MAP_layer+1) to num_MAP
       {DSL_node *mNode1 = net->GetNode(MAP_array[i]);
       mNode1->Value()->ClearEvidence();    //1. clear evidence
            net->SetTarget(MAP_array[i]);           //2. set target
            }
//4 update_belief
            net->UpdateBeliefs();
//5 order
            Order_single_MAP_variables_entropy(MAP_layer) ;
//6 extend_node
            Extend_node (MAP_layer );}
       return (best_sofar);
}
```


Figure 5.6: *double A_Star_Search(int num_of_MAP).*

Chapter 6

Experimental Results

6.1 Experimental Environment

To test the DWA^* algorithm, I studied its performance on many MAP problems in real Bayesian networks. I compare our results against those of current state of the art MAP algorithms: the P-Loc [14], P-Sys [15] and AnnealedMAP [21] algorithms respectively. I implemented the DWA^* algorithm in C++ and performed our tests on a 2.4 GHz Pentium IV Windows XP computer with 750MB memory.

The DWA^* algorithm is supported by the Structural Modeling, Inference, and Learning Engine (SMILE) which is developed by Decision Systems Laboratory (DSL).

6.2 Experimental Design

In order to exam the performance of the DWA^* algorithm comprehensively, I compared the efficiency and accuracy of the DWA^* algorithm with the other three existent algorithms P-Loc, P-Sys, AnnealedMAP in terms of running time and probability of the assignment. I used the same set of Bayesian networks that have been used for testing the performance of the MAP-solving algorithms at the Decision Systems Laboratory, University of Pittsburgh, and the Automated Reasoning Group, University of California, Los Angeles, which include Alarm [4], Barley [11], CPCS179 and CPCS360 [19], Diabetes [2], Hailfinder [1], Munin [3], Pathfinder [9], P223layout, and Win95pts [10], some of which are constructed for diagnosis. I also tested the algorithms on two very large proprietary diagnostic networks built at the HRL Laboratories (HRL1 and HRL2). The statistics for these networks are summarized in Table 6.1. I divided the networks into three groups: (1) small and middle-sized, (2) large but tractable, and (3) hard networks.

Table 6.1: Statistics for the Bayesian networks that I am using.

Group	Network	#Nodes	#Arcs
	Alarm	37	46
	CPCS179	179	239
	CPCS360	360	729
1	Hailfinder	56	66
	Pathfinder	135	195
	P223layout	223	338
	Win95pts	76	112
2	Munin	1,041	1,397
	HRL1	1,999	3,112
	HRL2	1,528	2,492
3	Barley	48	84
	Diabetes	413	602

For each network, I randomly generated 20 cases, and ran the above four algorithms on them. For each case, I randomly chose 20 MAP variables among the root nodes or all the them if root nodes were less than 20. I chose the same number of evidence nodes from among the leaf nodes. To set evidence, I sampled from the prior probability distribution of a Bayesian network in its topological order and cast the states of the sample to the evidence nodes. Following previous tests of MAP algorithms, I set the search time limit to be $3,000$ seconds (50 minutes). In all of our experiments, I used the default settings and parameters of P-Loc, P-Sys, AnnealedMAP unless mentioned specifically.

6.3 Results for the First and Second Group

In the first experiment, I ran the P-Loc, P-Sys, AnnealedMAP and DWA^* on all the networks in the first and second group, and all of the four algorithms generate results within the time limit. The P-Sys algorithm reported that it found all the optimal solutions. Table 6.2 reports the number of MAP problems that are solved correctly by the P-Loc AnnealedMAP and DWA^* algorithms. They all performed well on these networks. The DWA^* was able to find all the optimal solutions. The P-Loc algorithm missed only one case on the P223layout network and the AnnealedMAP missed one on Haifinder and two cases on P223layout.

Since both AnnealedMAP and P-Loc failed to find all the optimal solution in P223layout, in each of the 20 cases I studied the performance of the 4 algorithms as a function of the number of MAP variables (I randomly generated 20 cases for each number of MAP variables).

Because the search time of P-Sys increased very fast with the number of MAP variables, and it failed to generate any result when the number of MAP

Table 6.2: The number of cases that are solved correctly out of 20 random cases for the first and second group of networks.

	P-Loc	A-MAP	DWA^*
Alarm	20	20	20
CPCS179	20	20	20
CPCS360	20	20	20
Hailfinder	20	19	20
Pathfinder	20	20	20
P223layout	19	18	20
Win95pts	20	20	20
Munin	20	20	20
HRL1	20	20	20
HRL2	20	20	20

Table 6.3: The running time (in seconds) and the number of cases that the other 3 algorithms found smaller probabilities than DWA^* Search in network P223layout using their default settings.

MAP	P-Sys		P-Loc		A-MAP		DWA^*
	Runtime	Smaller	Runtime	Smaller	Runtime	Smaller	Runtime
10	0.265	0	0.361	0	1.575	0	0.221
20	23.236	0	1.179	1	12.089	2	2.385
30	68.829	0	2.563	1	32.579	0	9.923
40	TimeOut	-	3.305	4	10.601	4	5.906
50	TimeOut	-	4.219	6	12.168	2	10.578
60	TimeOut	-	5.031	5	15.481	2	10.112
70	TimeOut	-	5.906	6	15.981	5	10.312
80	TimeOut	-	6.828	6	11.171	1	11.093

variables reached 40, while the DWA^* Search found all the largest probabilities, I compared all the other 3 algorithms with DWA^* Search. With the increase of the number of MAP variables, both the P-Loc and AnnealedMAP turned to be less accurate for P223layout. When the number of MAP variables was above 40, there were about 25% cases of P-Loc and 15% cases in which AnnealedMAP found smaller probabilities than DWA^*.

Since only P-Loc spent less time than DWA^* when using its default settings, I am interested in the result when increasing the search steps of P-Loc such that it spends the same time as DWA^*. However, in practice the search time is not continuous in the number of search steps, so I just tried to find the parameters for P-Loc such that it spent only a little bit **more** time than DWA^*. Table. 6.4 shows the comparison between P-Loc and DWA^* in terms of run time and the number of cases that the two algorithms found different

result. We can see that after increasing the search steps of P-Loc, DWA^* still kept better accuracy when compared with P-Loc.

Table 6.4: The running time(in seconds) and the number of cases that the P-Loc found larger/smaller probabilities than DWA^* in network P223layout when spending a little bit more time than DWA^*.

MAP	P-Loc		DWA^*	
	RunTime	P-Loc $< DWA^*$	RunTime	P-Loc $> DWA^*$
10	0.262	0	0.181	0
20	3.685	0	3.531	0
30	8.134	0	7.150	0
40	8.140	1	6.635	0
50	8.221	2	6.792	0
60	8.215	2	7.248	1
70	9.968	3	8.599	2
80	11.609	5	9.520	0

In addition to the precision of the results, I also compared the efficiency of the algorithms. Table 6.5 reports the average running time of the four algorithms on the first and the second groups of networks. For the first group, the

Table 6.5: Average running time in seconds of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms on the first and second group of networks.

	P-Sys	P-Loc	A-MAP	A^*
Alarm	0.011	0.019	0.076	0.006
CPCS179	0.030	0.134	0.250	0.019
CPCS360	0.057	90.202	0.820	0.123
Hailfinder	3.910	0.118	0.452	0.239
Pathfinder	0.054	0.061	0.050	0.001
P223layout	32.370	1.376	12.166	2.507
Win95pts	0.031	0.041	0.292	0.030
Munin	3.382	5.353	19.620	2.996
HRL1	1.287	224.968	7.157	0.418
HRL2	0.087	5.45	4.071	0.384

AnnealedMAP, P-Loc and P-Sys algorithms showed similar efficiency on all except the CPCS360 and P223layout networks. The DWA^* search generated solutions within the shortest time on average. The small variance of the search time indicates that DWA^* is more stable across different networks.

For the second group, which consists of large Bayesian networks, P-Sys, AnnealedMAP and DWA^* are all efficient. DWA^* search still spent shortest search time on average, while the P-Loc was much slower on the HRL1 network.

6.4 Results for the Third Group

The third group consisted of two complex Bayesian networks: Barley and Diabetes, many nodes of which have more then 10 different states. As the P-Sys algorithm did not produce any results within the time limit, the only available measure of accuracy was a relative one: which of the algorithms found an assignment with higher probability. Table 6.6 lists the number of cases that were solved differently between P-Loc, AnnealedMAP, and the DWA^* algorithm and the number of cases that the DWA^* algorithm found a more probable assignment. P_L, P_A and P_* stand for the probability of MAP solutions found by P-Loc, AnnealedMAP and DWA^* respectively.

Table 6.6: The number of cases that were solved differently from P-Loc, AnnealedMAP and DWA^*.

	$P_* > P_L / P_* < P_L$	$P_* > P_A / P_* < P_A$
Barley	3/2	5/3
Diabetes	5/0	4/0

For Barley, the accuracy of the three algorithms is quite similar. However, for Diabetes DWA^* is more accurate: it found solutions with largest probabilities for all 20 cases, while P-Loc failed to find 5 and AnnealedMAP failed to find 4 of them.

Table 6.7: Average running time in seconds of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms on the third groups of Bayesian networks.

	P-Sys	P-Loc	A-MAP	A^*
Barley	TimeOut	101.47	34.67	199.16
Diabetes	TimeOut	369.35	315.79	185.89

DWA^* turned out to be slower than P-Loc and AnnealedMAP on Barley but more efficient on Diabetes (see Table 6.7).

6.5 Results for Incremental Evidence and MAP Variables Test

In order to give a more comprehensive comparison of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms, I run the four algorithms on Munin network in different number of evidence variables or different MAP variables. I chose the Munin network for this experiment because only this network has suitable number of root nodes and leaf nodes, 183 and 259 respectively, and I was able to run all four algorithms on it.

For the first step, I generated MAP problem with an increasing number of evidences nodes and keep the number of MAP nodes to be 50. The running times for each of the four algorithms are shown in Figure 6.1. We can see that the AnnealedMAP spent much longer time in generating the search result. The P-Sys, P-Loc, and DWA^* spent similar time when the number of evidence nodes is less than 100. But when the number of evidence nodes is beyond 100, the DWA^* is more efficient than P-Sys and P-Loc. The only exception happened when the number of evidences nodes was between 140 and 150.

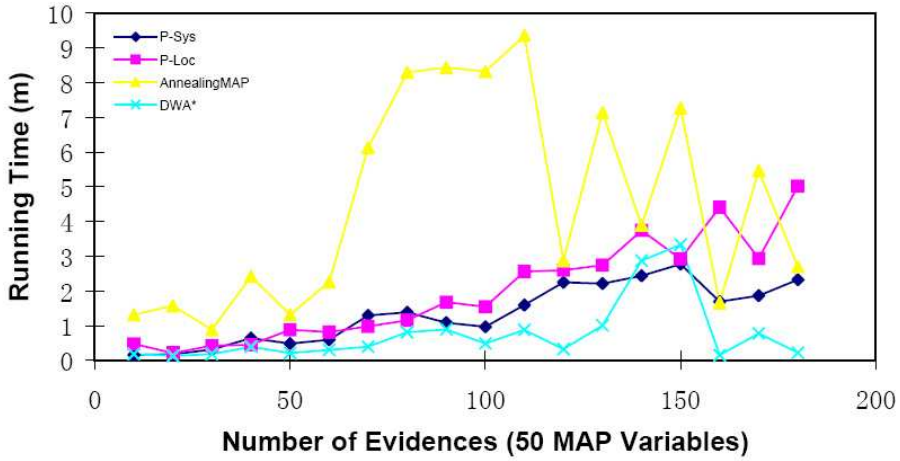

Figure 6.1: Plot of the running time of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms when increasing the number of evidence nodes on the Munin network.

For the second step, I generated MAP problem with an increasing number of MAP nodes and keep the number of Evidence nodes to be 50. The running times for each of the four algorithms are shown in Figure 6.2. It indicates that the P-Sys, P-Loc, and DWA^* algorithms were all very efficient when the number of MAP variables was not too large. However, when there were more MAP nodes which leads the MAP problem to be much harder, the AnnealedMAP and DWA^* were more efficient.

My last experiment focused on the robustness of the four algorithms to the number of nodes in the MAP set and the evidence set. In this experiment, I generated MAP problems with an increasing number of MAP and evidence nodes at the same time and ran four algorithms on these cases. The P-Sys was able to solve only cases with fewer than 140 MAP and evidence variables. The times for each of the cases are shown in Figure 6.3.

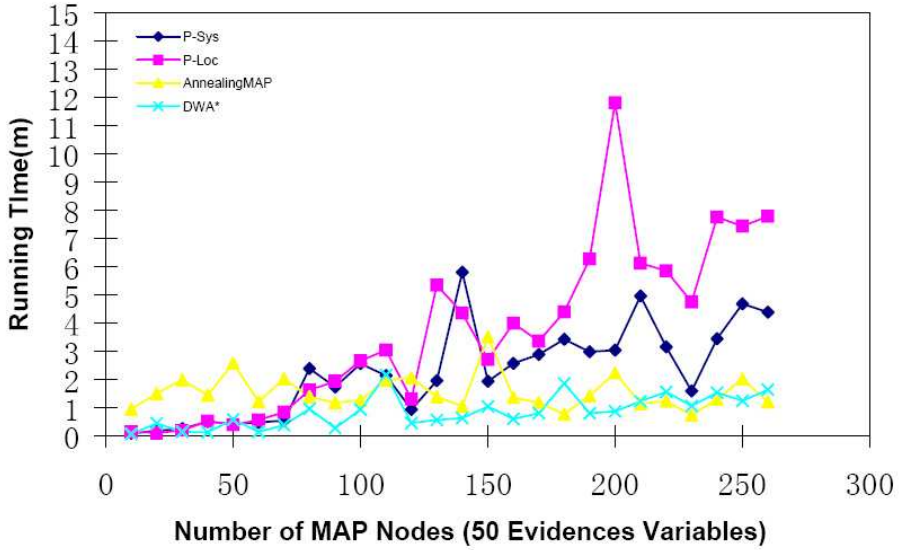

Figure 6.2: Plot of the running time of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms when increasing the number of MAP nodes on the Munin network.

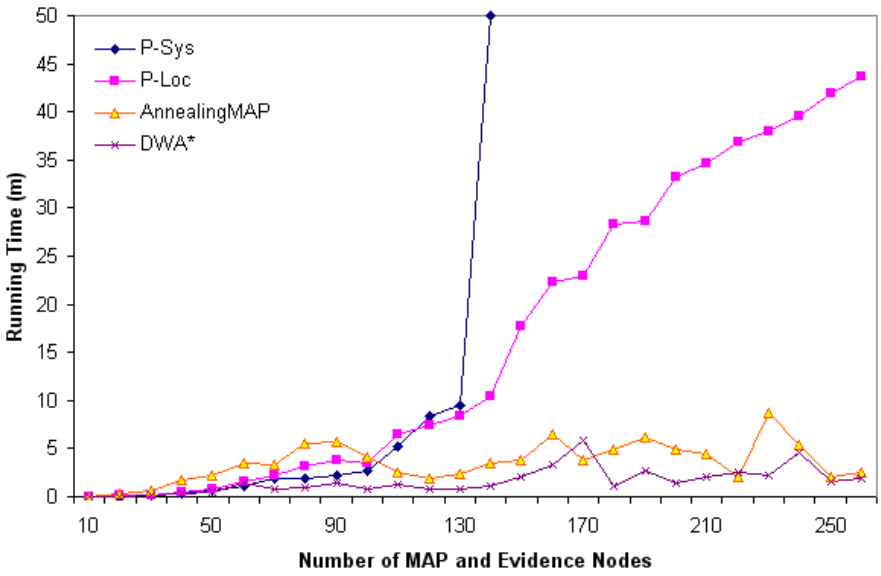

Figure 6.3: Plot of the running time of the P-Sys, P-Loc, AnnealedMAP and DWA^* algorithms when increasing the number of MAP nodes and evidence nodes at the same time on the Munin network.

Chapter 7

Conclusion

MAP problems in Bayesian networks are hard because they are not only subject to the complexity of the models (treewidth), but also subject to the complexity introduced by specific problems (constrained width).

My research on MAP problems at the Decision Systems Laboratory is mainly based on the theory of asymmetries among joint probability distributions. Although for the first three months, my tentative research of exploiting the estimation of the probability density functions was proved to be unsuccessful, and yet it drove me to realize that the accuracy of the upper bound on the MAP problem is the most influential element for the algorithm, which is quite sensitive to the error of the estimated mean and variance of the lognormal distribution function. Temporarily trapped in the darkness, I got new inspiration from the theory of asymmetries. That is to use greedy guess instead of estimating the lognormal probability density function in order to get a tight and accurate upper bound on MAP problem. The instructions from my supervisor Professor Druzdzel and positive comments on this idea from my colleague Changhe Yuan greatly encouraged me to embark on the implementation of the DWA^* algorithm, and lead to the accomplishment of this new efficient solution for MAP problems.

Another point that I would like to address is that the search algorithm that I chose. When I implemented the branch-and-bound search algorithm combined with the estimation of the lognormal probability density function, I found that the branch-and-bound was always turned to be a futile one: on one hand, when the upper bound is far larger than the probability of scenario, take 1.0 for example, the algorithm was far less efficient since there was not any "cut" in the probability tree; one the other hand, when the error of the estimation led the bound to be a lower bound instead of an upper one, the search path that could lead to the right solution would be cut by mistake. Then the branch-and-bound search would not generate the right scenario as the solution.

Compared with the branch-and-bound search, the Dynamic Weighting A^* Search is more robust for the MAP problems. Because of large asymmetries

among probabilities that are further amplified by their multiplicative combination, it is testified that for most of cases, the DWA^* can lead to the optimal solution, even though the heuristic function will sometimes break the rule of admissibility, if only the greedy guess is not too divergent from the ideal estimate, the algorithm will still not diverge from the optimal probability. Our simulation results also proved the robustness of the algorithm in finding optimal solutions.

The programming work of implementing the DWA^* search algorithm was accomplished by using C++ in the Windows programming environment with strong support of the SMILE library. The join tree algorithm and the relevance reasoning is an integral part of the SMILE library on which the implementation of my algorithm is based. I am very pleased that the DWA^* search algorithm will be included in the new version of SMILE released later.

Finding MAP in Bayesian networks is hard. By exploiting asymmetries among the probabilities of possible assignments properties of joint probability distributions among all the possible assignments, the Dynamic Weighting A^* Search is able to greatly reduce the search space and lead to efficient and accurate solution of the MAP problem. Our experimental result also show that generally, the Dynamic Weighting A^* Search is more efficient than the existent algorithms. Especially for large and complex Bayesian networks, when the exact algorithm fails to generate any result within a reasonable time, the Dynamic Weighting A^* Search can still provide accurate solutions efficiently.

Further extension of this research is to apply the Dynamic Weighting A^* Search algorithm to the K-MAP problem, which is to find k most probable assignments for MAP variables. It is very convenient for the DWA^* algorithm to achieve that, since after finding the most probable assignment the algorithm keeps all the candidate assignments in the search frontier. I can expect that the additional search time will be linear in k.

In sum, the Dynamic Weighting A^* Search algorithm enriches the approaches for solving MAP problem and extends the scope of MAP problems that can be solved.

Bibliography

[1] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. Winkler. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57–72, 1996.

[2] S. Andreassen, R. Hovorka, J. Benn, K. G. Olesen, and E. R. Carson. A model-based approach to insulin adjustment. In M. Stefanelli, A. Hasman, M. Fieschi, and J. Talmon, editors, *Proceedings of the Third Conference on Artificial Intelligence in Medicine*, pages 239–248. Springer-Verlag, 1991.

[3] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck, and F. Jensen. MUNIN — an expert EMG assistant. In John E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam, 1989.

[4] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *In Proc. 2'nd European Conf. on AI and Medicine*, pages 38:247–256, Springer-Verlag, Berlin, 1989.

[5] L. de Campos, J. Gamez, and S. Moral. Partial abductive inference in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, 20(11-13):1211–1217, 1999.

[6] R. Dechter and I. Rish. Mini-buckets: A general scheme for approximating inference. *Journal of ACM*, 50(2):1–61, 2003.

[7] M. J. Druzdzel. Some properties of joint probability distributions. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 187–194, Morgan Kaufmann Publishers San Francisco, California, 1994.

[8] M. J. Druzdzel and H. J. Suermondt. Relevance in probabilistic models: "Backyards" in a "small world". In *Working notes of the AAAI–1994 Fall Symposium Series: Relevance*, pages 60–63, New Orleans, LA (An extended version of this paper is in preparation.), 4–6 November 1994.

[9] D. Heckerman. Probabilistic similarity networks. *Networks*, 20(5):607–636, August 1990.

[10] D. Heckerman, J. Breese, and K. Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38:49–57, 1995.

[11] K. Kristensen and I.A. Rasmussen. The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33:197–217, 2002.

[12] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)*, 50(2):157–224, 1988.

[13] J. D. Park. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI–02)*, pages 388–396, Morgan Kaufmann Publishers San Francisco, California, 2002.

[14] J. D. Park and A. Darwiche. Approximating MAP using local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI–01)*, pages 403–410, Morgan Kaufmann Publishers San Francisco, California, 2001.

[15] J. D. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI–03)*, pages 459–468, Morgan Kaufmann Publishers San Francisco, California, 2003.

[16] J. Pearl. *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley Publishing Company, Inc., 1988.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[18] S.Russell P.Norvig. *Artificial Intelligence A Modern Approach*. Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 1995.

[19] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 484–490, San Mateo, CA, 1994. Morgan Kaufmann Publishers, Inc.

[20] S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[21] C. Yuan, T. Lu, and M. J. Druzdzel. Annealed MAP. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI–04)*, pages 628–635, AUAI Press, Arlington, Virginia, 2004.