University of Castilla-La Mancha

A publication of the

Department of Computer Science

Searching for the best elimination sequence in Bayesian networks by using Ant Colony based Optimization

by

José A. Gámez and José M. Puerta

Technical Report **#DIAB-01-04-13** January 2000

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Searching for the best elimination sequence in Bayesian networks by using Ant Colony based Optimization

José A. Gámez and José M. Puerta

January 23, 2001

Abstract

The knowledge base of a probabilistic expert system is usually represented as a Bayesian network. Most of the knowledge engineering tools used in the development of probabilistic expert systems do not carry out the inference process directly over the network, but in a secondary graphical structure called a *junction tree*. The efficiency of inference (propagation) algorithms depends on the size of the junction tree obtained, and this size depends on the *elimination sequence* used during the compilation/transformation of the Bayesian network into a junction tree. The problem of searching for the best elimination sequence is a $\mathcal{NP}$-hard problem [Wen, 1991], and this has motivated the proliferation of approximate methods to approach it (based variously on heuristics, genetic algorithms, simulated annealing, etc.). In this paper we investigate the aplicability to this problem of a new combinatorial optimization technique, inspired by a natural model, which has appeared recently: the *ant colony-based optimization* [Dorigo and Gambardella, 1997]. Our approach is validated by using a set of complex networks obtained from a repository.

1 Introduction

Probabilistic methods were for some time discarded as a tool for dealing with uncertain reasoning, because they required too complex a specification and computation. Nevertheless, with the appearance of probabilistic network models (mainly Bayesian and Markov networks [Pearl, 1988, Castillo et al., 1997]), probability is enjoying a spectacular revival, being nowadays one of the most accepted and applied measures of uncertainty.

In most cases, the kernel of a *Probabilistic Expert System* (PES) is a set of Bayesian networks (usually one), because they provide an efficient representation of the joint probability distribution, and allow us to update probabilities by means of *local* computation, i.e., probabilistic computations are carried out over the initial pieces of information instead of using a global distribution.

A *Bayesian network* (BN) [Pearl, 1988, Jensen, 1996], also known as Bayesian (belief) network, causal network, or causal probabilistic network, is a directed acyclic graph (DAG),

$\mathcal{G} = (X_{\mathcal{U}}, \mathcal{E})$, where each node represents a random variable (having a finite number of possible states) and the links indicate some kind of direct relationship (usually causal connections) between the linked variables. The quantitative part of the model expresses the strength of the connections (links) between the problem variables, and is given by a probability distribution for each node conditioned to its parents. If $X_{\mathcal{U}} = \{X_1, \ldots, X_n\}$ is the set of variables in the network, then due to the relations of independence induced by the topology of the graph, the joint probability can be factorized as:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | pa(X_i)), \tag{1}$$

where $pa(X_i)$ contains the parents of X_i in $\mathcal{G}$.

Evidently, this factorization of the joint probability distribution make the task of knowledge acquisition and representation easier, because the number of parameters required is much smaller. As an example, let us consider the network in Fig. 2-a, assuming that all the variables have two states that we need to acquire (and store) 36 values, instead of $2^8 = 256$, which would be required in order to specify the joint probability distribution.

Once the model has been specified our aim is to use it in order to perform some kind of reasoning. The two types of reasoning more commonly used in PES's are *predictive* reasoning, also known as *probabilities or evidence propagation* [Pearl, 1986, Andersen et al., 1989], and *abductive* reasoning, also known as *diagnostic* reasoning [Pearl, 1987, Gelsema, 1995, de Campos et al., 1999]. Although there are some methods that directly work over the network [Pearl, 1986, Pearl, 1987, Gelsema, 1995], nowadays the most frequently used propagation algorithms are based on the *compilation* (transformation) of the BBN in a secondary structure called a *junction tree* [Andersen et al., 1989, Shenoy and Shafer, 1990, Nilsson, 1998, de Campos et al., 2000]. Junction tree based algorithms are more general in the sense that they are able to deal with every network topology.

A junction tree is a higher level graphical structure (a tree) whose nodes are subsets (non-disjointed) of the variables in the network and in which some properties must hold. As different junction trees can be obtained from the same network, our goal should be to obtain the one that best suits the propagation task. However, the problem of searching for that *optimal* junction tree is $\mathcal{NP}$-hard [Wen, 1991]. Because of this, several approaches have been proposed in order to search for close-to-optimal junction trees, the following two approaches being those that have received most attention in the specific BN literature:

- *Heuristic algorithms* [Kjærulff, 1990, Cano and Moral, 1994]. This type of method provides a *good* solution quickly.

- *Algorithms based on Combinatorial Optimization techniques* [Larrañaga et al., 1997, Kjærulff, 1992, Hernández, 1995]. This type of method provides high-quality solutions, better (in general) than those achieved by the heuristic approach, although the amount of CPU time required is considerably greater.

In our opinion, both types of algorithms are really useful, although in different stages of the probabilistic expert system development. In fact, we can distinguish two different situations in which the knowledge engineer performs the task of *compilation*:

- During the construction of the *knowledge base*, i.e., the Bayesian network. At this stage of the process, the knowledge engineer (KE) builds the network(s) in the *edit window* and tests the results of the modifications carried out in the *run window*[1]. Due to the fact that each time the KE switches from the edit window to the run window, a compilation occurs, and due also to the large number of modifications carried out by the KE over the network before the final version is obtained, is obvious the need for a fast compilation process. Heuristic algorithms are therefore particularly suitable at this stage. We will refer to compilation at this stage as *on-line* compilation.

- Before producing the final product to be given to the user. At this stage, our goal should be to produce as good a junction tree as possible, because hundreds or thousands of propagations will be carried out over it. Therefore, at this stage we can spend more time on the compilation process, in order to achieve a better junction tree, and so algorithms based on combinatorial optimization techniques are suitable. We will refer to compilation at this stage as *off-line* compilation.

  As an example of the importance of having a good junction tree, let us consider the plot shown in Fig. 1. In this plot the CPU time required by the propagation task[2] (Y axe) is shown when junction trees of different sizes (X axe) are considered for the same *complex* network.

[1] This is the typical way of working in the most popular knowledge engineering tools oriented toward the development of PES's, as HUGIN (`www.hugin.dk`) or NETTICA (`www.norsys.com`)

[2] Propagation has been carried out by using HUGIN [Andersen et al., 1989] algorithm implemented in Java language (virtual machine 1.2) and run over a Pentium III 500 MHz computer running Linux operating system

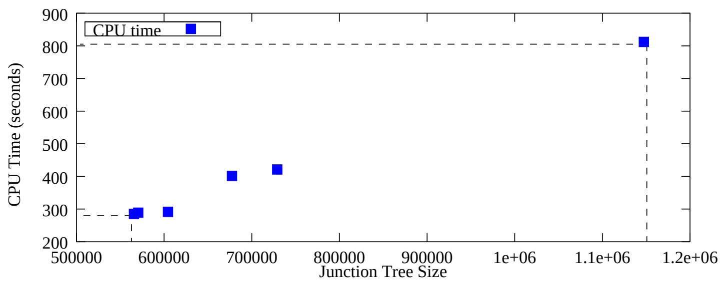

Figure 1: CPU time comparison for different junction tree sizes

In this paper we propose a novel way of facing the problem of searching for a close-to-optimal junction tree, by making use of Ant Colony Optimization (ACO) algorithms. ACO [Bonabeu et al., 1999, Dorigo and Gambardella, 1997] is a new paradigm of bio-inspired algorithms that has shown a very good behaviour when it is applied to solving hard combinatorial optimization problems as the *Traveling Salesman Problem* or the *Quadratic Assignment Problem*.

The algorithm here proposed integrates the two previously cited paradigms, because ACO falls within the combinatorial optimization strategies, but each ant will use (during the construction of its solution) the knowledge provided by a concrete heuristic. Furthermore, due to the advantage of guiding the global search by heuristic knowledge, we hope to obtain a high quality solution faster than the rest of combinatorial optimization methods (as genetic algorithms).

The rest of the paper is structured as follows. Section 2 provides an introduction to the graph triangulation problem, which is the key point in the compilation process. In Section 3, the proposed algorithm is presented, by giving all the necessary steps to define an ACO algorithm for the graph triangulation problem. In Section 4, the behaviour of the proposed algorithm is tested by means of experimental evaluation. Finally, in Section 5, some concluding remarks are made. A brief introduction to the Ant Colony Optimization paradigm is presented in Appendix A.

2 The graph triangulation problem

The process of building a junction tree from a Bayesian network, also known as *compilation*, contains the following steps:

1. *Moralization.* The moral graph is obtained by adding undirected links between all the common parents of a node, and then dropping the direction of all the links presented in the initial graph (network). As an example, Fig. 2-b shows the moralization of the graph depicted in Fig. 2-a, which corresponds to the famous *Asia* network introduced in [Lauritzen and Spiegelhalter, 1988].

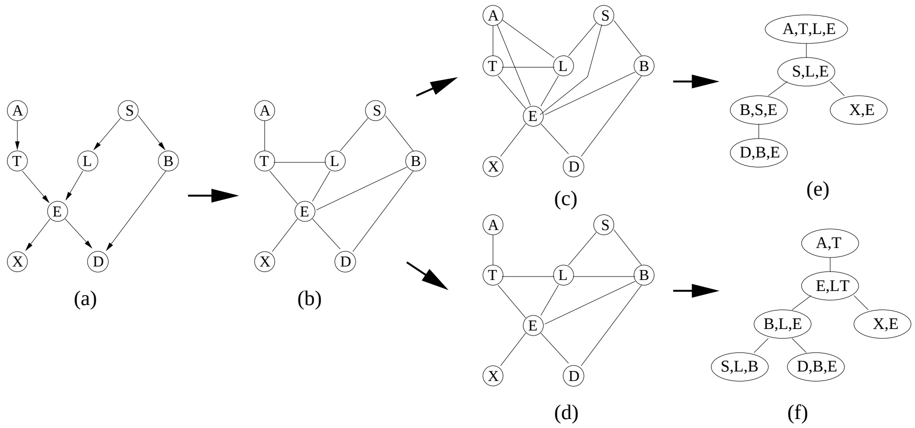

Figure 2: From a network to a junction tree

2. *Triangulation* of the moral graph. An undirected graph is said to be *triangulated* or *chordal* if any cycle of length greater than 3 has a chord. Therefore, a triangulated graph can be obtained from an undirected graph by adding new links. To do so, a permutation (usually known as the *deletion sequence* and denoted by σ) of the nodes in the graph ($\{X_1, \ldots, X_n\}$) is chosen, and the following (elimination) process is carried out:

   (a) For $i = 1$ to n do

      a.1 Let X_i to be the variable in position i of sequence σ.

      a.2 Add links between all nodes adjacent to X_i ($adj(X_i)$) until $\{X_i \cup adj(X_i)\}$ becomes complete, that is, all the nodes are pairwise adjacent. Let T_i be the set of added links.

      a.3 Remove X_i and all their incidents links from the graph.

   (b) The triangulated graph is obtained by adding the set of links $\{T_1, \ldots, T_n\}$ to the original undirected graph.

   Figure 3 shows the elimination process for the moral graph shown in Fig. 2-b driven by the elimination sequence $\{T, A, X, S, D, L, B, E\}$.

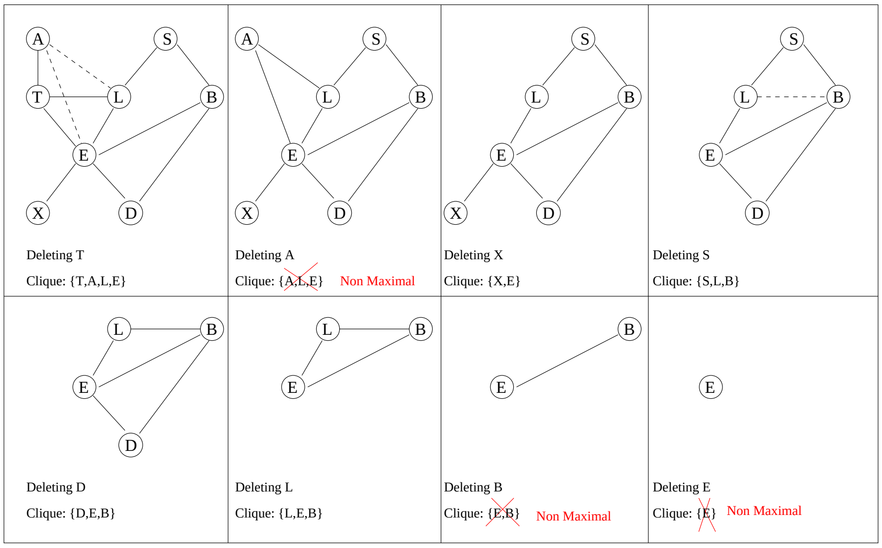

Figure 3: Elimination process. Dashed lines are those added during the process

3. Get the *cliques* (maximal complete sub-graphs) of the triangulated graph. As we can see in Fig. 3 the cliques can be identified during the elimination process.

4. .....

Because the size (state space size) of a junction tree only depends on the size of the cliques it contains, the remaining steps (arranging the cliques in a tree-shaped structure and probabilistic representation) fall outside the scope of our study (see [Jensen, 1996] for details). Therefore, we have only carried out the symbolic (graphical) part of the process, and not the numerical part, which is by far the most time consuming.

From the process described above, we can conclude that obtaining a triangulated graph is not a problem, because all the steps are deterministic except the selection of the elimination sequence, althoug we can select *any* sequence. However, we should not forget that our aim is to use the triangulated graph in order to obtain a junction tree to be used for probabilistic propagation. Hence, our goal is to obtain triangulations yielding small cliques, since the size of the probability tables attached to each clique depends on the size of the clique itself, and the complexity of propagation algorithms depends on the size of the probability tables to be used.

In a more formal way, if C_i is a clique, $\mathcal{T} = \{C_1, \ldots, C_t\}$ is a junction tree, $s(C_i)$ is the size of C_i (calculated as the product of the number of states of the variables contained in C_i), and $s(\mathcal{T})$ is the size of $\mathcal{T}$ (calculated as the sum of the sizes of each one of its cliques), then our goal is to obtain a triangulated graph yielding a junction tree of the smallest possible size.

In our example (Fig. 2), parts (c) and (d) show two possible triangulations of (b), while parts (e) and (f) show the junction trees obtained from the graphs (c) and (d) respectively. As we can see, the junction tree depicted in (e) is more complex, because it has size 44 if all the variables have 2 states (size 171 in the case of 3 states), while the junction tree depicted in (f) has size 40 (126).

Although the triangulation process is simple, the process of searching for an optimal triangulation (deletion sequence) is $\mathcal{NP}$-hard [Wen, 1991]. However, several heuristic algorithms have been proposed in order to obtain *good* deletion sequences in a quick way. Below, we describe three of the most commonly used in the literature (each one relying on a different criterion):

- *minFill* [Kjærulff, 1990]. In each step, select the variable X which minimizes the number of links to be added when X is deleted. We will denote this value for variable X as $H_F(X)$.

- *minVars*[3] [Kjærulff, 1990]. In each step, select the variable X which minimizes the number of variables to be included in the clique obtained when X is deleted. We will denote this value for variable X as $H_V(X)$.

- *CanoMoral*[4] [Cano and Moral, 1994]. In each step, select the variable X which minimizes the value $s(C_i)/s(X)$, where C_i is the clique produced by the deletion of variable X, and $s(X)$ is the number of possible states for X. We will denote this value for variable X as $H_{CM}(X)$.

As no tie-breaking is provided, when two or more variables have the same value $H_{F,V,CM}(\cdot)$, one of them is randomly selected.

[3] Although Kjærulff called this heuristic *minSize*, we refer to it as minVars because the *word* size will be used only in relation to state space size.

[4] This is an improved version of heuristic *minWeight* proposed by Kjærulff, and in which the criterion to be minimized is $log_2(s(C_i))$.

3 Applying ACO algorithms to the graph triangulation problem

Solving the graph triangulation problem by means of combinatorial optimization techniques consists in the searching of the best elimination sequence in the search space defined by the set of all the possible permutations of the variables in the network (graph). Depending on the type of technique, the solution is obtained in a different way. Thus, in *simulated annealing* [Kjærulff, 1992] the algorithm changes the current sequence by permuting some of the sequence members, while in *genetic algorithms* [Larrañaga et al., 1997, Hernández, 1995] the new solutions arise from the re-combination of solutions previously seen in past populations.

None of the previous methods make use of heuristic knowledge related to the problem, which is one of the main characteristics of the ACO algorithms, and helps to focus upon (and speed up) the search process.

In the following subsections we describe the design of the proposed ACO algorithm, by detailing each one of its components (with some of them we make a comparison with the well known *Traveling Salesman Problem* (TSP) in order to clarify their meaning).

3.1 Problem Representation

In order to apply ACO algorithms, our problem must fulfill the two following conditions:

1. It should be viewed as a combinatorial optimization problem.

2. It should have the capability of being represented on a graph.

As we have seen before, the graph triangulation problem can be represented as a combinatorial one, by searching for the best elimination sequence in the search space of all the possible permutations of the nodes in the graph. Thus, condition (1) is clearly fulfilled.

The representation of the problem on a graph could look trivial, because we are dealing with a graph (the network). However, we cannot use the graph which represents the network to be triangulated as the graphical representation, because not all the permutations can be obtained as paths (cycles) in this graph. Because of this, the graph to be considered will be a complete graph defined over the variables in the network, in this way, it is always possible to reach node i from node j for every pair of nodes (i, j). We thus have a graph representation equal to the one used for the TSP problem, although in its asymmetric case, because, in general, it is not equal to delete X_i before than X_j, that the other way around.

3.1.1 Reducing the cardinality of the search space

With the previous problem definition, it is clear that the search space cardinality is $n!$, n being the number of nodes (variables) in the network. However, it is possible to reduce (drastically in some cases) this search space. To do so, we are going to apply the *reduction* operation introduced by [Hernández, 1995].

If $G(X_i)$ is the sub-graph defined by $\{X_i\} \cup adj(X_i)$, then the idea of the reduction operation is to remove first those nodes such that $G(X_i)$ is *complete* in the moral graph. Hernández shows that there is no better candidate to be eliminated at each step, than the one fulfilling the previous condition.

The input of the reduction operation will be the moral graph $\mathcal{G}$, and the output will be:

- σ_r: the sequence containing the nodes eliminated during the reduction process.
- $\mathcal{L}_r$: the list of cliques obtained by the deletion of the nodes in σ_r.
- $s(\sigma_r)$: the sum of the sizes of the cliques in $\mathcal{L}_r$.
- $\mathcal{G}_r$: the reduced graph, that is, the graph obtained from $\mathcal{G}$ by removing the nodes in σ_r and all their incidents links. Notice, that there are no new links in $\mathcal{G}_r$, because $G(X_i)$ is complete in $\mathcal{G}$ for every $X_i \in \sigma_r$.

Our goal now is to obtain the elimination sequence σ' for the graph $\mathcal{G}_r$. If this process yields σ', $\mathcal{L}'$, and $s(\sigma')$, then the data for the initial graph $\mathcal{G}$ can be obtained as:

- $\sigma = \sigma_r \cdot \sigma'$, with $\cdot$ being the concatenation operator.
- $s(\sigma) = s(\sigma_r) + s(\sigma')$.
- $\mathcal{L} = \mathcal{L}_\nabla \cup \mathcal{L}'$.

If n is the number of nodes in $\mathcal{G}$, and n' is the number of nodes in $\mathcal{G}_r$, then, the search space cardinality has been reduced from $n!$ to $n'!$. As an example, if reduction is applied to the moral graph shown in Fig. 2-b, we obtain $\sigma_r = (D, A, T, X)$, and $\mathcal{G}_r = (\{E, L, S, B\}, \{(E, L), (L, S), (S, B), (B, E)\})$. Thus, the search space cardinality has been reduced from $8! = 40,320$ to $4! = 24$.

3.2 Heuristic Knowledge

In ACO algorithms (App. A) the array of pheromone constitutes a dynamic structure whose content is adaptive in the sense that it varies with time, representing learning by memory. On the other hand, the heuristic knowledge is usually static, in the sense that it is calculated before any ant is launched. As an example, in the TSP the heuristic knowledge to be considered when studying the possibility of going from city i to city j (ν_{ij}), is calculated as the the inverse of the distance between cities i and j.

Nevertheless, this is not the case in the graph triangulation problem, in which the heuristic information is also dynamic. In fact, in the problem here considered, the heuristic knowledge $H_{F,V,CM}(X_j)$ depends on the nodes previously deleted. Thus, the value $\nu_{ij} = f(H_{F,V,CM}(X_j))$ needs to be calculated on-line, that is, just after deleting the node X_i.

To solve this problem we propose the following solution: each ant implements one of the available heuristics, that is, each ant carries out a triangulation over its own copy of the graph. In this way, the array of pheromone will be a global structure, while the heuristic knowledge will be local to each ant.

As our problem consists in a minimization (as in the TSP), we will consider the following heuristic information:

- Ants implementing heuristic H_F: $\nu_{ij} = \frac{1}{1+H_F(X_j)}$. We have added 1 to the value returned by the heuristic because this value could be 0.
- Ants implementing heuristic H_V: $\nu_{ij} = \frac{1}{H_V(X_j)}$.
- Ants implementing heuristic H_{CM}: $\nu_{ij} = \frac{1}{1+log_2(H_{CM}(X_j))}$. We have take log_2 because the values returned by H_{CM} may be too large in complex networks $(10^5, 10^6, ...)$.

3.3 Fitness Function

Although our goal is to minimize $s(\sigma)$, it is clear that after applying reduction, this problem is equivalent to minimizes $s(\sigma')$, because $s(\sigma) = s(\sigma_r) + s(\sigma')$, and $s(\sigma_r)$ is always the same value.

In other techniques as genetic algorithms, this value can be obtained by triangulating a copy of $\mathcal{G}'$ using σ', and then calculating the size of the cliques obtained. However, given that in our case each ant will perform a triangulation of the graph in order to obtain the heuristic knowledge at each step, it is clear that the ant can return not only the sequence, but also the size associated with that sequence. Thus, no posterior evaluation of the sequence is required.

3.4 Pheromone Initialization

Initially, a small amount of pheromone, τ_0, is deposited in every link of the graph. This amount is usually calculated as a function of the goodness of a solution to the problem obtained using a quick method (a greedy algorithm). In our case, the initial amount of pheromone is calculated as:

$$\tau_0 = \frac{1}{n \cdot s(\sigma')},$$

where n is the number of variables in $\mathcal{G}_r$ and σ' is a deletion sequence obtained by applying the same heuristic that will be implemented by *all* the ants in the colony, or by applying H_V if different types of ants are present in the colony, that is, if there are ants implementing different heuristics, included in the colony.

3.5 ACO Algorithm

In order to decide which ACO algorithm should be applied to our problem, we have implemented the two approaches described in appendix A: Ant System (AS) and Ant Colony System (ACS). After some trials we realized that ACS suits our problem better than AS (which requires much many iterations), and so we decided to pursue our research only with this method. So, Eq. 5 is used as a *transition rule* during the solution construction process carried out by an ant.

The other important factor in the design of an ACO algorithm is the way in which the pheromone will be updated. Following the classical implementation of ACS [Dorigo and Gambardella, 1997] two types of pheromone updating will be carried out:

- *Global pheromone trail update rule.* After each iteration, the links belonging to the best solution found from the beginning of the search, σ^+, are reinforced with an extra amount of pheromone. At the same time, evaporation of pheromone in every link is performed by using a parameter ρ. The global update rule is:

  $$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}, \tag{2}$$

  where $\{ij\}$ are the links in σ^+, ρ is the parameter associated with pheromone evaporation, and $\Delta\tau_{ij} = \frac{1}{s(\sigma^+)}$. Notice, that this procedure helps to focus the search around the neighbourhood of σ^+.

- *Local pheromone trail update rule.* This updating is carried out by the ants, when they are building their solutions. Each time an ant k moves from node i to node j, it updates the pheromone level in link (i, j) by applying the following expression:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0. \tag{3}$$

  Some other parameters instead of τ_0 have been used to carry out the local pheromene update, but although the alternatives required greater computational effort, as reported in [Bonabeu et al., 1999], no significant improvement in the behaviour of the algorithm was obtained.

  Therefore, each time a link is visited, its pheromone level diminishes, allowing the exploration of links not yet visited, and so avoiding the premature convergence problem. Without this local update rule, all the ants in the colony will be looking around the best path (solution) found to date, impeding other solutions from emerging.

3.6 Other aspects

In this subsection we comment upon some other relevant aspects of our algorithm.

- As suggested in [Dorigo and Gambardella, 1997], a *candidate list* of length cl will be considered. That is to say, if the ant is in node i, only the nodes included in the candidate list will be considered in the probabilistic transition rule. This approach aids in speeding up the process, since it minimizes the number of computations to be carried out when applying the transition rule.

  In our case, the candidate list will be integrated by the cl nodes which minimize the value $H_{F,V,CM}(\cdot)$. This list can be obtained in a very efficient way, because each ant implements a priority queue with the node candidates to be deleted in the next step, and as the priority is given by the inverse of $H_{F,V,CM}(\cdot)$, then we only have to cut the list of nodes to be considered at position cl.

- Another interesting point worth commenting upon is the selection of the first node. In problems like the TSP, the election of the first city it of minor importance, because the goal is to obtain a cycle. In our problem, however, the election of the first node to be deleted is very important. In fact, if we make a bad selection, the first clique could be too large, resulting in a bad junction tree.

In our case, we have adopted a slightly complex method to select the first node of σ'.

- In each iteration a small percentage (10-20%) of the ants select the node to be considered as origin from a set containing those nodes which minimize some of the heuristic considered.
- The rest of ants select their origin using the following procedure:
  1. Generate a random uniform number $u \in [0, 1]$
  2. If $u <= p$ select the origin randomly
  3. If $u > p$ select the origin with probability proportional to the goodness of the best sequence obtained with X_i as origin, $i = 1..n$.

  Where p is a number in $[0, 1]$ whose value decreases with time, allowing more exploration in the first iterations. In our case $p = 0.5$ during the first 25% of the iterations, and $p = 0.2$ in the remaining iterations.

4 Experimental Evaluation

We have tested our algorithm with four *complex*[5] Bayesian networks obtained from a repository[6] maintained by the research group in *Decision Support Systems* at Aalborg University. Associated to each network, a *good* deletion sequence for the network can also be downloaded.

Table 1 shows some information about the networks used in experimentation, including the size of the junction tree obtained when using the downloaded sequence, and the effect of applying the reduction operation. As we can see, the cardinality of the search space is reduced drastically in all the cases.

For the purpose of analyzing the behaviour of the proposed ACO algorithm, we will compare it with the three heuristic algorithms detailed in Section 2, and with a genetic algorithm (GA) based on the results presented in [Larrañaga et al., 1997], which was proved to have a similar behaviour to the simulated annealing algorithm presented in [Kjærulff, 1992]. The structure of the genetic algorithm is shown below, and is based on the steady state evolutionary mechanism. The combination of CX [Oliver et al., 1987] as crossover operator and ISM [Michalewicz, 1992] for mutation, was found to produce the best results by Larrañaga

[5]It is usual to denominate in this way those networks which yield large size junction trees
[6]`http://www.cs.auc.dk/research/DSS/misc.hml`

Table 1: Information about the networks used in experimentation (#n = number of nodes, #l = number of links, $\mathcal{G}$ = moral graph, $\mathcal{G}_r$ = reduced graph, $s(P(X_i))$ = size of the probability table attached to X_i, s(Aalborg) = size of the junction tree obtained when using the downloaded sequence)

	$\mathcal{G}$		$s(P(X_i))$			$\mathcal{G}_r$		s(Aalborg)
Network	$\#n$	$\#l$	*min*	*max*	*mean*	$\#n$	$\#l$	
Water	32	123	3	3,072	421.4	24	101	8,035,356
Mildew	35	80	3	280,000	15,633.1	20	40	4,179,856
Barley	48	126	2	40,320	2,172.1	35	92	17,338,949
Munin1	189	366	2	600	102.9	108	241	188,387,156

et al. The selection mechanism is based on the rank of the individuals according to their fitness.

Structure of the genetic algorithm

1 Create initial population

2 For $i = 1$ to numberOfGenerations do

2.1 Select two parents p_1 and p_2 according to their rank

2.2 $ch_1 = CX(p_1, p_2)$

2.3 $ch_2 = CX(p_2, p_1)$

2.4 Generate two random uniform numbers $u_1, u_2 \in [0, 1]$

2.5 If $u_1 <=$ probabilityOfMutation then $ch_1 = ISM(ch_1)$

2.6 If $u_2 <=$ probabilityOfMutation then $ch_2 = ISM(ch_2)$

2.7 Enlarge the population with ch_1 and ch_2

2.8 Remove the worst two individuals of the population

3 Return the best individual found

The genetic algorithm has been carried out over the reduced graph, so it works over the same search space as the ACO algorithms. Although in [Larrañaga et al., 1997, Hernández, 1995] the initialization of the population is carried out randomly, we have considered two cases: (a) random initial population, and (b) when half of the population is initialized randomly and the other half is initialized by using the heuristic algorithms. In this way, we establish some good starting points in the initial population. Finally, the population size has been fixed to be twice the number of nodes in the reduced graph, the number of generations is based on the population size, and the probability of mutation has been fixed (by experimentation) to 0.3 (Table 6 shows the concrete parameters for each experiment).

With respect to the ACO approach, we have run eight versions of the basic one, which result from the following combinations:

1. The kind of ants used, that is, the heuristic implemented by each ant of the population. We have considered four options:

   - ACS_F: All the ants implement heuristic H_F.
   - ACS_V: All the ants implement heuristic H_V.
   - ACS_{CM}: All the ants implement heuristic H_{CM}.
   - ACS_{all}: 1/3 of the ants implement H_F, 1/3 of the ants implement H_V, and 1/3 of the ants implement H_{CM}.

2. Two population sizes have been considered.

   - Population of size 10. This number was found to be adequate for a large variety of problems in [Dorigo and Gambardella, 1997].
   - Population of size 0.25$*$ numberOfNodes. We have taken this number and not the number of nodes, because as mentioned above, not all the nodes are good candidates to be considered as the origin of the sequence.

   Thus, the ACO algorithm will be identified by the algorithm implemented (Ant Colony System), the kind of ants included in the population, and the population size. For example, $ACS_F - 10$.

The number of iterations is calculated in such a way that the maximum number of evaluated individuals is the same for both population sizes. The remaining parameters (shown in Table 6) have been obtained after a hard experimentation, in which we have tried out different values for $q_0(0.5, 0.6, 0.7, 0.8$ and $0.9)$, $\rho(0.1, 0.2, 0.3, 0.4$ and $0.5)$ and $\beta(0.5, 1, 2$ and $5)$, while the size of the candidate list was 15 in the first three networks (as in [Dorigo and Gambardella, 1997] for the TSO), and 30 for the Munin1 network. As we can see in table 6, the selection of parameters q_0 and β are those which make extensive use of the available knowledge during the search process. With respect to the evaporation parameter, it seems that as we have limited the number of iterations to be carried out, ρ increases (at least when 10 ants per population are considered) with the difficulty of the problem, so that, the search is focused faster around promising paths.

The results are shown in Tables 2 to 5. All the algorithms have been carried out 50 times over each network. The best results are shown in boldface. In these tables,

- *Alg* identifies the used algorithm,
- *best* stands for the size of the best sequence obtained in the 50 runs,
- A stands for the average size of the 50 runs,
- SD stands for the standard deviation, and
- EBS for the number (averaged over the 50 runs) of evaluated individuals until the best solution was found.

Table 2: Results obtained for the network *Water* (over 50 runs)

Alg.	*best*	*A*	*SD*	*EBS*
H_F	3,657,180	3,657,180	**0.0**	–
H_V	6,746,844	7,228,253	468,045	–
H_{CM}	8,035,356	8,035,356	**0.0**	–
ACS_F-10	**3,028,305**	3,099,232	135,498	213
ACS_F-6	**3,028,305**	**3,095,589**	134,738	259
ACS_V-10	**3,028,305**	3,443,565	190,106	306
ACS_V-6	3,126,609	3,417,405	102,577	300
ACS_{CM}-10	3,362,268	3,443,737	*77,428*	297
ACS_{CM}-6	3,362,268	3,471,899	200,255	240
ACS_{all}-10	**3,028,305**	3,207,461	158,521	258
ACS_{all}-6	**3,028,305**	3,297,447	134,054	**202**
GA-random	**3,028,305**	3,754,198	1.0286e6	2,614
GA-half	**3,028,305**	3,295,475	134,941	1,233

4.1 Analysis

The analysis is performed from three different angles:

Accuracy. The results obtained lead us to highlight the good behaviour of the ACO algorithms, as can be deduced from the following sentences:

- In all the networks except in Barley at least one ACO algorithm has obtained a solution as good as the one obtained by the genetic algorithms, and in the case of Barley, solutions very close to the best have been found by three of the ACO algorithms.

Table 3: Results obtained for the network *Mildew* (over 50 runs)

Alg.	*best*	*A*	*SD*	*EBS*
H_F	4,296,900	12,202,198	6.6228e6	–
H_V	3,440,016	44,228,847	6.4466e7	–
H_{CM}	4,584,208	4,584,208	**0.0**	–
ACS_F-10	**3,400,464**	3,437,691	51,575	259
ACS_F-5	**3,400,464**	3,426,748	31,109	257
ACS_V-10	**3,400,464**	3,433,821	55,084	269
ACS_V-5	**3,400,464**	3,688,640	568,364	295
ACS_{CM}-10	**3,400,464**	**3,400,662**	*1,118*	118
ACS_{CM}-5	**3,400,464**	3,403,238	13,246	**108**
ACS_{all}-10	**3,400,464**	3,440,989	77,606	246
ACS_{all}-5	**3,400,464**	3,497,974	195,332	228
GA-random	**3,400,464**	4,201,658	1.7003e6	1,960
GA-half	**3,400,464**	3,424,670	20,035	467

Table 4: Results obtained for the network *Barley* (over 50 runs)

Alg.	*best*	*A*	*SD*	*EBS*
H_F	17,287,568	49,342,994	5.1636e7	–
H_V	17,296,011	76,049,483	6.9074e7	–
H_{CM}	19,807,811	19,807,811	**0.0**	–
ACS_F-10	**17,140,796**	**17,144,846**	*12,247*	488
ACS_F-8	**17,140,796**	17,154,261	23,587	389
ACS_V-10	17,189,796	18,101,920	1.9180e6	560
ACS_V-8	**17,140,796**	18,782,038	2.8384e6	562
ACS_{CM}-10	17,198,036	17,291,056	85,130	469
ACS_{CM}-8	**17,140,796**	17,307,284	165,300	**356**
ACS_{all}-10	**17,140,796**	17,167,039	38,691	462
ACS_{all}-8	**17,140,796**	17,194,279	84,475	419
GA-random	**17,140,796**	21,625,419	9.1661e6	8,237
GA-half	**17,140,796**	17,214,307	119,043	5,074

Table 5: Results obtained for the network *Munin1* (over 50 runs)

Alg.	*best*	*A*	*SD*	*EBS*
H_F	201,504,306	598,631,341	6.1416e8	–
H_V	183,476,315	404,186,469	2.7737e8	–
H_{CM}	195,365,325	241,792,763	4.6895e7	–
ACS_F-10	83,846,417	**92,945,493**	8.7506e6	1,937
ACS_F-27	87,610,584	102,940,075	1.1743e7	2,043
ACS_V-10	83,835,810	102,323,200	1.2829e7	**1,765**
ACS_V-27	87,021,730	98,061,360	9.4292e6	1,896
ACS_{CM}-10	87,052,504	97,844,460	6.0324e6	1,856
ACS_{CM}-27	83,775,680	97,011,683	**5.4973e6**	1,931
ACS_{all}-10	87,057,806	98,681,051	1.0730e7	1,778
ACS_{all}-27	86,255,780	99,985,103	1.0133e7	1,968
GA-random	83,740,888	423,501,803,437	2.561e12	43,119
GA-half	**83,735,918**	117,508,857	3.0303e7	28,540

Table 6: Parameters considered in the experiments

network	*it*	*ants*	ρ	β	q_0	*cl*	*pop*	*gen*	*pm*
Water	60	10	0.2	5	0.6	15	48	4,800	0.3
Water	100	6	0.5	5	0.7	15			
Mildew	50	10	0.4	5	0.7	15	40	4,000	0.3
Mildew	100	5	0.3	5	0.8	15			
Barley	80	10	0.4	5	0.8	15	70	7,000	0.3
Barley	100	8	0.4	5	0.9	15			
Munin1	270	10	0.5	5	0.7	30	216	21,600	0.3
Munin1	100	27	0.4	5	0.9	30			

- With respect to standard deviation, and leaving aside the heuristic methods, we can see that in all the cases the smallest SD corresponds to an ACO algorithm. However, a more detailed analysis shows us that with respect to SD, GA-half has a similar behaviour to ACO algorithms, being clearly superior to GA-random.

- With respect to average, in all the cases the best result is obtained by an ACO algorithm (usually which also has the smallest standard deviation). In order to ascertain whether there are significant differences among the algorithms used, we have carried out a statistical analysis (t-test), in which a 95% confidence level has been used for the analysis of two samples. Two ACO algorithms have been selected for this analysis, namely, those yielding the best average in each case, and $ACS_{all} - 10$ as a common representative for all cases. From the results of the statistical study the following conclusions can be obtained:

  - Statistically significant differences have been found between the heuristic method with best average and ACO algorithms.
  - Statistically significant differences have been found between the heuristic method with best average and GA-half, but not with respect to GA-random (notice that we have fixed a maximum number of generations, while in [Larrañaga et al., 1997] the GA go on until the population converges).
  - In almost all cases, significant differences have been found between the two ACO algorithms studied and GAs.
  - In three of the four cases there is significant difference between GA-half and GA-random.
  - In all cases there is a significant difference between the two ACO algorithms studied.

Efficiency It is in this aspect in which ACO algorithms stand out. In fact, taking the number of evaluated individuals before the best solution is found as a measure, the ACO algorithms found the best solution between 5 and 16 times quicker than GA-half and between 13 and 24 times quicker than GA-random. This is due to the use of heuristic information that guides the ACO algorithms in the search process. Furthermore, as each ant returns the fitness of its constructed solution, no posterior evaluation therefore being required, the time employed by an ant and that needed to build a solution (crossover + mutation?) and evaluate it in the genetic algorithm, is similar.

Another interesting point emerging from the results, is the fact that the difference grows with the network complexity.

Which of the proposed ACO algorithms should be used? From the results it seems that a population of 10 ants is a good choice, as suggested in [Dorigo and Gambardella, 1997]. With respect to the kind of ants to be introduced in the population, we can make the following observations:

- As we can see, the kind of ants implemented by the ACO algorithm with the best average, does not coincide in all the cases with the best heuristic algorithm. However, from the results, it seems that *minFill* is the heuristic yielding the best results (three of the four considered networks).

- It seems that there is no need to introduce different kinds of ants in the same population, because in none of the cases does this option improve the behaviour of ACS_F.

It appears, therefore, that $\text{ACS}_F - 10$ is the best option among the proposed algorithms, at least for the kind of networks examined here.

5 Concluding Remarks

In this work, a novel way of dealing with the graph triangulation problem has been proposed. It is centered around the application of Ant Colony based Optimization, which differs from other combinatorial optimization techniques in as much as the available heuristic knowledge is used during the construction of potential solutions. As has been demonstrated, the use of this knowledge has improved the results obtained by other techniques, both in accuracy and efficiency.

With respect to the genetic algorithm used in order to test the ACO algorithms, it is clear that the introduction in the initial population of good solutions obtained by heuristic algorithms, helps to speed up the search process.

As further work, we propose to apply new ACO approaches to the problem. We also plan to study how to scale the problem to larger networks, for which some sort of network division would clearly be necessary in order to deal with smaller search spaces.

A Ant Colony Optimization Algorithms

Ant Colony Optimization algorithms (ACO) [Bonabeu et al., 1999] are multi-agent systems in which the behaviour of each agent (*ant*) is inspired by the foraging behaviour of real ants. In particular, ACO algorithms model the process followed by real ants when looking for a short path between a food source and the nest.

The way in which a short path is built by the colony is based on the ants' ability to deposit and sniff a chemical substance called *pheromone*. In fact, when an ant is going from the nest to the food source and vice versa, it deposits a small amount of pheromone on the ground. The amount of pheromone in a path is used to guide the colony during the search, so that, when an ant is located in a branch and has to take a decision, it makes a probabilistic decision biased by the amount of pheromone deposited on the different branches.

In this way, at the beginning of the process, all the paths have the same probability of being chosen (because there is no pheromone at all), but with the continuous action of the colony, the shortest paths are more frequently visited, receiving a higher amount of pheromone and thereby becoming more attractive for the subsequent ants. By contrast, the longest paths are less visited and this together with the pheromone evaporation process, makes these paths less attractive for subsequent ants. Hence, the final solution (path) emerges from the *colaboration* among all the members in the colony.

Although ACO algorithms were initially used to solve problems related to path searching in graphs, mainly the *Traveling Salesman Problem* (TSP) [Dorigo and Gambardella, 1997], in a more recent study, Dorigo and Di Caro (1999) introduce a *Meta-heuristic* based on the ACO approach, which is valid for dealing with general optimization problems if they can be represented as a graph.

Pheromone is modeled by means of a bi-dimensional array τ of dimension $n \times n$, where n is the number of nodes in the graph. τ_{ij} contains the pheromone amount deposited in the *direct* path which connects node i with node j. Thus, in the former *Ant Systems* (AS) [Dorigo et al., 1996], when an ant k is located at node i, it chooses node j as the following node to be visited with a probability proportional to:

$$p_k(i,j) = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\nu_{ij}]^\beta}{\sum_{u \in J_k(i)} [\tau_{iu}]^\alpha \cdot [\nu_{iu}]^\beta} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where:

- ν_{ij} represents *heuristic* information (knowledge) about the problem. As an example, in the TSP $\nu_{ij} = \frac{1}{d_{ij}}$, with d_{ij} being the distance between nodes i and j.

- $J_k(i)$ is the set of nodes for which there is a direct path from node i and not yet visited by ant k.

- α and β are two parameters used to control the relative importance of pheromone with respect to heuristic information.

Subsequently, some modifications were carried out over the original AS in order to improve its performance. Perhaps the more important of them was introduced in [Dorigo and Gambardella, 19 and consists in the modification of the transition rule. In the new version, known as *Ant Colony System* (ACS) it is possible to decide explicitly the balance between *exploration* of new edges and *exploitation* of heuristic knowledge. To do so, a new parameter $0 \leq q_0 \leq 1$ is introduced, and the following transition rule is applied:

$$j = \begin{cases} \arg\max_{u \in J_k(i)} \left\{ [\tau_{iu}] \cdot [\nu_{iu}]^{\beta} \right\} & \text{if } q \leq q_0 \\ J & \text{if } q > q_0 \end{cases} \tag{5}$$

where q is a random number uniformly distributed in [0,1], and $J \in J_k(i)$ is a node selected according to Eq. (4) with $\alpha = 1$.

At each iteration of the algorithm, each ant builds (progressively) a new solution to the problem by means of repetitive application of the probabilistic transition rule. At the end of each iteration, the pheromone trails are updated by:

- *Evaporation*: by decreasing the amount of pheromone in all the edges by some constant factor.

- *Reinforcement*: by depositing an amount of pheromone in the edges belonging to the generated solutions. The amount of pheromone to be deposited depends on the quality of the constructed solution, the better the solution, the greater the amount of pheromone deposited.

Depending on the way the degrees of freedom of the previous idea are interpreted, different *pheromone update rules* emerge, which give rise to different ACO models (see [Bonabeu et al., 1999], chapter 2).

References

[Andersen et al., 1989] Andersen, S., Olesen, K., Jensen, F., and Jensen, F. (1989). Hugin: a shell for buiding belief universes for expert systems. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit.

[Bonabeu et al., 1999] Bonabeu, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Ingelligence: From Natural to Artificial Systems* Oxford University Press, Oxford, UK.

[Cano and Moral, 1994] Cano, A. and Moral, S. (1994). Heuristic algorithms for the triangulation of graphs. In *Proceedings of the 5th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU), Vol. 1*, pages 166–171, Paris (France).

[Castillo et al., 1997] Castillo, E., Gutiérrez, J., and Hadi, A. (1997). *Expert Systems and Probabilistic Network Models.* Monographs in Computer Science. Springer-Verlag, New York.

[de Campos et al., 1999] de Campos, L., Gámez, J., and Moral, S. (1999). Partial Abductive Inference in Bayesian Belief Networks using a Genetic Algorithm. *Pattern Recognition Letters*, 20(11-13):1211–1217.

[de Campos et al., 2000] de Campos, L., Gámez, J., and Moral, S. (2000). On the problem of performing exact partial abductive inference in Bayesian belief networks using junction trees. In *Proceedings of the 8th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU), Vol. 3*, pages 1270–1277, Madrid (Spain).

[Dorigo and Di Caro, 1999] Dorigo, M. and Di Caro, G. (1999). The Ant Colony Optimization Meta-Heuristic, in *New Ideas in Optimization.* McGraw-Hill.

[Dorigo and Gambardella, 1997] Dorigo, M. and Gambardella, L. (1997). Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Trans. on Evolutionary Computation*, 1:53–66.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 26:29–41.

[Gelsema, 1995] Gelsema, E. (1995). Abductive reasoning in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, 16:865–871.

[Hernández, 1995] Hernández, L. (1995). *Diseño y validación de nuevos algoritmos para el tratamiento de grafos de dependencias*. PhD thesis, Dpto. de Ciencias de la Computación e I.A. Universidad de Granada.

[Jensen, 1996] Jensen, F. (1996). *An introduction to Bayesian Networks*. UCL Press.

[Kjærulff, 1990] Kjærulff, U. (1990). Triangulation of graphs - algorithms giving small total space. Technical Report R 90-09, Department of Mathematics and Computer Science. Institute of Electronic Systems. Aalborg University.

[Kjærulff, 1992] Kjærulff, U. (1992). Optimal decomposition of probabilistic networks by simulated annealing. *Statistic and Computing*, 2:7–17.

[Larrañaga et al., 1997] Larrañaga, P., Kuijpers, C., Poza, M., and Murga, R. (1997). Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing*, 7:19–34.

[Lauritzen and Spiegelhalter, 1988] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J.R. Statistics Society. Serie B*, 50(2):157–224, 1988.

[Oliver et al., 1987] Oliver, I.M., Smith, D.J., and Holland, J.R. (1987). A study of permutation crossover operators on the TSP. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*. Cambridge, MA, pages 224–230.

[Michalewicz, 1992] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin.

[Nilsson, 1998] Nilsson, D. (1998). An efficient algorithm for finding the M most probable configurations in Bayesian networks. *Statistics and Computing*, 2:159–173.

[Pearl, 1986] Pearl, J. (1986). Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29:241–288.

[Pearl, 1987] Pearl, J. (1987). Distributed revision of composite beliefs. *Artificial Intelligence*, 33:173–215.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo.

[Shenoy and Shafer, 1990] Shenoy, P. and Shafer, G. (1990). Axioms for probability and belief-function propagation. In Shachter, R., Levitt, T., Kanal, L., and Lemmer, J., editors, *Uncertainty in Artificial Intelligence, 4.*, pages 169–198. Elsevier Science Publishers B.V. (North-Holland).

[Wen, 1991] Wen, W. (1991). Optimal decomposition of belief networks. In Bonissone, P., Henrion, M., Kanal, L., and Lemmer, J., editors, *Uncertainty in Artificial Intelligence 6*, pages 209–224. North-Holland.