Leiden University
Leiden Institute of Advanced Computer Science
Mathematical Institute of Leiden University

Inference in Markov Networks

Willem Obbens

Supervised by

Siegfried Nijssen

Johannes Schmidt-Hieber

Submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science in Computer Science and Mathematics, 26 August 2014.

Abstract

This thesis covers methods for performing exact as well as approximate inference in Markov and Bayesian networks. Specifically, we study MPE inference where the problem is to find states of probability distributions with maximal probability. As Markov networks are strictly more general than Bayesian networks, the majority of the algorithms in this thesis are designed for the former type of network. We propose a methodology consisting of three components: ant colony optimisation, belief propagation and integer linear programming.

In previous research, [Dar09] considered variable elimination and a branch-and-bound depth first search algorithm on the assignment tree to compute exact solutions to the problem of MPE inference. Furthermore, [GBH04] created an ant colony algorithm to find approximate solutions.

Finding an exact solution to this problem is NP-complete in general [Dar09], so exact MPE inference is intractable in the case of big networks. In this scenario, one would normally resort to approximation algorithms. However, sometimes an exact solution is required, such as when important decisions have to be made depending on the solution. In that respect it is also useful to be able to quickly compute an approximate solution and use it as a lower bound constraint to compute an exact solution more quickly. The problem is that the current exact algorithms have no way to easily integrate such a constraint into a problem. As a solution, this thesis provides a reduction of the problem of MPE inference in Markov networks to integer linear programming (ILP) problems, where it is straightforward to add a lower bound constraint on the objective function. There is also a wealth of literature on ILP solvers and several decades of work, something we can benefit from greatly [Nie].

As for approximation algorithms, we designed a hybrid belief propagation/ant colony algorithm for MPE inference in Markov networks and a pure ant colony algorithm for MPE inference in Bayesian networks, inspired by [GBH04]. The hybrid algorithm is exponential in the maximum clique size and its core performs belief propagation on a spanning tree which is generated by the ant colony spanning tree search component. The pure ant colony algorithm is linear in the number of network variables.

For benchmarking, the algorithms were programmed in the non-strict functional programming language Haskell. The experiments, using datasets from the Probabilistic Inference Challenge 2011, show that neither of these two algorithms outperforms the other and that they perform approximately equally fast on small datasets. They also show that solutions found by the approximation algorithms can decrease the execution time of the ILP solver if the approximation is used as a lower bound constraint.

Keywords: MPE inference, Markov network, Bayesian network, belief propagation, ant colony optimisation, integer linear programming.

Acknowledgements

First and foremost, I am indebted to my supervisors Siegfried Nijssen and Johannes Schmidt-Hieber as many ideas in this thesis were proposed by them. Despite my father's passing away on July 16th, 2014, his presence in my mind gave me the strength to complete this thesis. I am also grateful for the encouragement by my mother who was always there for me and helped me in my life. Finally, I would like to thank the creator of this very flexible LaTeX template.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we give an introduction to the problem addressed in this thesis. We will first explain what Markov networks are and subsequently what inference means in the context of Markov networks.

Historically, Markov networks were originally used by physicists to analyse interactions of lattices. An important theorem providing sufficient and necessary conditions for probability distributions to be represented as Markov networks was proved by [HC71] in 1971.

Later, in 1985, Pearl introduced Bayesian networks as a model for self-activated memory. Bayesian networks are able to represent causal links between random variables and have various applications in diagnostics.

1.1 What is a Markov network?

A Markov network is a graphical way of representing dependency relations between random variables and their joint probability distribution. Intuitively, it is a way of cutting a probability distribution into smaller more approachable pieces. The network is a hypergraph where the nodes are random variables and hyperedges are groups of variables which are called cliques. For each clique there exists a unique factor which is a real-valued function from the variables in the concerned clique. The joint probability distribution of the random variables in the network is given by the product of all the factors. It follows that every random variable must be a member of at least one clique, as the joint probability distribution would not depend on it otherwise. In principal all Markov network distributions are assumed to be strictly positive unless specified otherwise, due to the Clifford-Hammersley theorem by [HC71]. The theorem states that a probability distribution can be factored as a product of factors from a Markov network iff the probability distribution is strictly positive, i.e. its image is $\mathbf{R}_{>0}$.

In this thesis we mainly consider Markov networks, which are more general than Bayesian networks as any Bayesian network can be converted into a Markov network.

1.2 What is inference?

Inference comprises several generic operations for extracting information from probability distributions, such as maximisation or marginalisation. This thesis goes deeper into the former, specifically MPE inference which is maximisation over all the variables in a network.

1.3 Applications

In general the algorithms in this thesis could be used in classification problems. However, some concrete applications of this thesis' achievements include:

- **Natural language processing:** Semantic role labelling (SRL); find the most likely grammatical meaning of every word in a sentence. In this application the variables of the network could by represented by the words in the sentences and the values of the variables are given by all possible grammatical meanings of a word. Hence, all variables have the same value set.

  A simple example, with the three categories "indefinite article", "noun" and "verb":

A	thesis	is	being	written.
verb	verb	**verb**	**verb**	**verb**
indef. article	indef. article	indef. article	indef. article	indef. article
noun	**noun**	noun	noun	noun

  The correct grammatical meanings of the words are printed in **bold**.

- **Object recognition in images:** Highlight the pixels in an image which have the highest probability of belonging to a certain object.

- **Speech recognition:** Given recordings of the same word by different people, link the correct person to future recordings.

1.4 Related work

Bayesian networks were first defined by Judea Pearl in [Pea85] in 1981, as a model for evidential reasoning. However, the more general Markov networks had already been described before by Clifford and Hammersley in [HC71] as early as 1971, mainly motivated as a model that physicists could use for interactions on lattices.

In his first work on Bayesian networks, Pearl describes the belief propagation algorithm for Bayesian networks with a tree structure, an algorithm that was later generalised to Markov networks that possibly contain loops (see, for example, [YFW01]).

Keeping in mind the fact that we could easily add constraints to integer linear programs, we proceeded our research in the direction of approximation algorithms. The solution of this type of algorithm can be used as a lower bound in an integer linear program and speed up the execution of the ILP solver.

An ant colony MPE algorithm (ANT-MPE) for Bayesian networks was described by [GBH04] and supplied with a new approach and explanation in this thesis; the approach used in this thesis can be seen as modelling MPE inference as a Markov model.

We also researched belief propagation in Markov networks (such as defined in [YFW01]) and managed to amalgamate it with an ant colony algorithm into the BP-ACO algorithm. This combination was motivated by the probabilistic nature of ant colony algorithms and the success of the belief propagation. The result is an algorithm which is slightly probabilistic in order to search for a good message propagation order, but still keeps the good traits from the belief propagation algorithm.

1.5 Contributions

The main contributions of this thesis are:

- A formulation of MPE inference as an integer linear programming problem. Being able to view the problem of MPE inference as an ILP problem is useful, because we can add different sorts of constraints to an ILP problem. We could for example add an approximate solution as a lower bound constraint, which would allow the ILP solver to prune more branches, causing a significant speed-up in cases of big datasets. Furthermore, if we have multiple networks over the same variables and want to perform MPE inference in one of them while we have some constraints on the probability distributions of the other networks, then those constraints can be easily added to the ILP problem.

  For example, given three networks over the variables X and Y, this sort of optimisation problem could

look like

$$\begin{aligned} &\max \mathcal{P}_1(X, Y) \\ &\text{subject to } \mathcal{P}_2(X, Y) \leq \frac{1}{2} \text{ and } \mathcal{P}_3(X, Y) \geq \frac{1}{3}. \end{aligned}$$

In this thesis, we mention how this can be done but do not consider it further.

- A new approach for an ant colony procedure to approximate MPE instantiations. The ant colony algorithm from this thesis was based on the algorithm from [GBH04]. The main difference between the two algorithms is that the version of this thesis uses a data structure that makes the algorithm a bit more easily understandable.

- A generalisation of the state-of-the-art belief propagation algorithm. The standard belief propagation method was modified to maximise a distribution instead of marginalise it and generalised to arbitrary not necessarily tree-shaped networks by choosing a spanning tree with an ant colony algorithm and executing the normal belief propagation algorithm on the spanning tree.

  This algorithm should not be confused with loopy belief propagation which is also a generalisation of normal belief propagation to networks without a tree structure.

1.6 Further remarks

In this thesis, we will only consider finitely-valued (discrete) probability distributions induced by the networks. Hence, when statistical definitions are given that hold for discrete and continuous distributions alike in general, the definition is tacitly assumed to be for discrete distributions and only the $\sum$-variant will be given, not the $\int$-variant.

Chapter 2

Background theory

2.1 Introduction

In this chapter some basic notation is defined that is necessary to read this thesis. This is also the chapter where the important concepts of Markov and Bayesian networks are introduced which lie at the heart of this thesis. The operations and predicates in this section should be familiar to the reader; this section serves merely as an aid to the reader to disambiguate notation in case of confusion.

The reader is further assumed to be acquainted with the predicate logical vocabulary, i.e. universal quantification ($\forall$), existential quantification ($\exists$), logical conjunction and disjunction ($\wedge$, $\vee$, resp.), etc.

2.2 Notation

2.2.1 Unary operators on sets

Let A be a set. The cardinality (number of elements for finite sets) of A is defined to be $\#A$.

If $A = \{a_1, \ldots, a_n\}$ is a set containing numbers, then

$$\prod A = a_1 \cdot a_2 \cdot \cdots \cdot a_n, \qquad \sum A = a_1 + a_2 + \cdots + a_n.$$

If A contains sets, then

$$\prod A = a_1 \times a_2 \times \cdots \times a_n, \qquad \bigcup A = a_1 \cup a_2 \cup \cdots \cup a_n, \qquad \bigcap A = a_1 \cap a_2 \cap \cdots \cap a_n.$$

2.2.2 Functions

Let $f\colon A \to B$ be a function, then

$$\operatorname{im}(f) = f(A) = \{f(a)\colon a \in A\}, \qquad \operatorname{dom}(f) = A, \qquad \operatorname{cod}(f) = B.$$

If B possesses a linear ordering, then

$$\arg\max_{a\in A} f(a) = \{a \in A\colon \forall a' \in A\,(f(a') \leq f(a))\}.$$

The operation arg min is defined similarly.

2.2.3 Linear algebra

Let $m, n \in \mathbf{N}$ and $\mathbf{K}$ a field, then the set $\operatorname{Mat}_{m\times n}(\mathbf{K})$ denotes the $m \times n$-matrices over $\mathbf{K}$. As a shorthand, we define $\operatorname{Mat}_n(\mathbf{K}) = \operatorname{Mat}_{n\times n}(\mathbf{K})$.

2.2.4 Approximation

Let $f, g\colon X \to \mathbf{K}$ be two functions and $\mathbf{K}$ a field, then

$$f \propto g$$

is equivalent to

$$\exists c \in \mathbf{K}\colon f = c \cdot g \quad \text{i.e.} \quad \exists c \in \mathbf{K}\, \forall x \in X\colon f(x) = c \cdot g(x).$$

In this thesis this equivalence relation will be used exclusively for "equality up to normalisation". This means that, if $P\colon X \to [0,1]$ is a probability distribution and $f\colon X \to \mathbf{R}$ then $P \propto f$ or $P(x) \propto f(x)$ means that $P(x) = \frac{1}{\sum_{y\in X} f(y)} f(x)$ for all $x \in X$. Finally, note that $\propto$ is an equivalence relation.

Big-O/Ω/Θ Notation

Let $f, g\colon X \to \mathbf{R}$ for some linearly ordered set X. Then

$$\begin{aligned} f \in \mathcal{O}(g) &\iff \exists x' \in X\, \exists K \in \mathbf{R}\, \forall x > x'\colon f(x) \leq K \cdot g(x), \\ f \in \Omega(g) &\iff \exists x' \in X\, \exists K \in \mathbf{R}\, \forall x > x'\colon f(x) \geq K \cdot g(x), \\ f \in \Theta(g) &\iff \exists x' \in X\, \exists K, L \in \mathbf{R}\, \forall x > x'\colon L \cdot g(x) \leq f(x) \leq K \cdot g(x). \end{aligned}$$

It should be noted that $f \in \mathcal{O}(g)$ if $f \propto g$.

2.2.5 General

If p is a logical formula, then $p \equiv_\alpha p'$ means that p' and p are α-equivalent, which means that they are equal up to renaming of bound variables.

Example 1.

$$\forall \alpha\colon \alpha \wedge \beta \equiv_\alpha \forall \gamma\colon \gamma \wedge \beta$$

because they are both of the form $\forall x\colon x \wedge \beta$ where β is a free variable, but

$$\exists \alpha\colon \alpha \vee \zeta \not\equiv_\alpha \exists \alpha\colon \alpha \vee \psi$$

because $\zeta \neq \psi$ in general. ■

2.3 Definitions

Definition 2.1. A Bayesian network B is a finite directed acyclic graph (DAG) with vertices $\mathcal{V}$ and edges $\mathcal{E}$. Every vertex $X \in \mathcal{V}$ is a finitely-valued variable, with its values represented by $\Omega(X)$ and its parents by $\operatorname{par}(X) = \{Y \in \mathcal{V}\colon (Y, X) \in \mathcal{E}\}$. Furthermore, X is also associated with a conditional probability table (CPT)

$$\mathcal{P}(X \mid \operatorname{par}(X)) : \Omega(X) \times \prod \Omega(\operatorname{par}(X)) \to \mathbf{R}_{\geq 0}.$$

The set of CPTs will be referred to with the symbol Θ.

The network induces a joint probability distribution $\mathcal{P}$, given by

$$\mathcal{P}(X_1, \ldots, X_n) = \prod_{k=1}^{n} \mathcal{P}(X_k \mid \operatorname{par}(X_k)),$$

where $n = \#\mathcal{V}$. ■

Example 2. This example shows a Bayesian network with 4 binary(-valued) variables A, B, C and D. A has no parents, B and C both have A as their parent and D has both B and C as its parents.

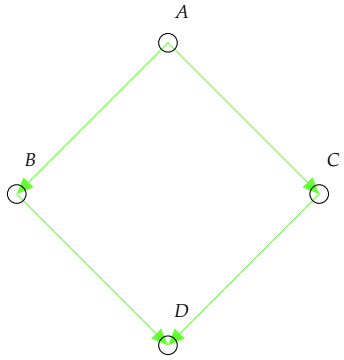


Figure 2.1: An illustrative Bayesian network.

It has the following CPTs:

A	$\mathcal{P}(B=0)$	$\mathcal{P}(B=1)$
0	$\frac{1}{2}$	$\frac{1}{2}$
1	$\frac{2}{3}$	$\frac{1}{3}$

$\mathcal{P}(A=0)$	$\mathcal{P}(A=1)$
$\frac{2}{3}$	$\frac{1}{3}$

A	$\mathcal{P}(C=0)$	$\mathcal{P}(C=1)$
0	$\frac{3}{4}$	$\frac{1}{4}$
1	$\frac{5}{7}$	$\frac{2}{7}$

B	C	$\mathcal{P}(D=0)$	$\mathcal{P}(D=1)$
0	0	$\frac{1}{3}$	$\frac{2}{3}$
0	1	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{7}{23}$	$\frac{16}{23}$
1	1	$\frac{4}{7}$	$\frac{3}{7}$

■

Definition 2.2. A Markov network M is a finite undirected graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$. Every vertex $X \in \mathcal{V}$ is a finitely-valued variable, with its values represented by $\Omega(X)$. The network also contains cliques, denoted by $\mathcal{C}$. A clique is a subset of the nodes in the network. An edge between two variables indicates that they appear in at least one clique together. Every clique $C \in \mathcal{C}$ corresponds to a factor φ_C defined by

$$\varphi_C \colon \prod \Omega(C) \to \mathbf{R}_{\geq 0}.$$

The set of factors is denoted by Φ. The network induces a joint probability distribution $\mathcal{P}$, given by

$$\mathcal{P}(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \varphi_C(X_i \colon X_i \in C),$$

where Z is a normalisation constant defined by

$$Z = \sum_{x \in \prod \Omega(V)} \prod_{C \in \mathcal{C}} \varphi_C(x_C),$$

where x_C is the vector of all C-components of x. ■

Note that in Bayesian networks, the CPTs are in 1-to-1 correspondence with the variables, but in Markov networks, the factors are in 1-to-1 correspondence with the cliques.

Example 3. This example shows a Markov network with 4 binary variables A, B, C and D. It isomorphic to the Bayesian network in example 2 and has the cliques $\mathcal{C} = \{\{A\}, \{A, B\}, \{A, C\}, \{B, C, D\}\}$.

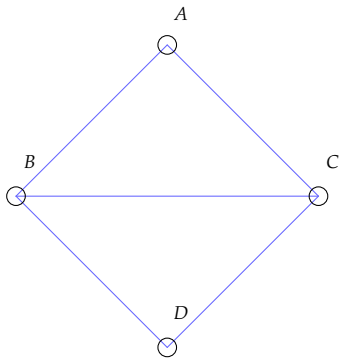


Figure 2.3: An illustrative Markov network.

It has the following factors:

A	B	$\varphi(A,B)$
0	0	$\frac{1}{2}$
0	1	$\frac{1}{2}$
1	0	$\frac{2}{3}$
1	1	$\frac{1}{3}$

A	$\varphi(A)$
0	$\frac{2}{3}$
1	$\frac{1}{3}$

A	C	$\varphi(A,C)$
0	0	$\frac{3}{4}$
0	1	$\frac{1}{4}$
1	0	$\frac{5}{7}$
1	1	$\frac{2}{7}$

B	C	D	$\varphi(B,C,D)$
0	0	0	$\frac{1}{3}$
0	0	1	$\frac{2}{3}$
0	1	0	$\frac{1}{2}$
0	1	1	$\frac{1}{2}$
1	0	0	$\frac{7}{23}$
1	0	1	$\frac{16}{23}$
1	1	0	$\frac{4}{7}$
1	1	1	$\frac{3}{7}$

■

Definition 2.3. The MORAL GRAPH of a directed graph G is an undirected graph $\mathbf{M}(G)$ such that $\mathcal{V}(\mathbf{M}(G)) = \mathcal{V}(G)$ and $\{X, Y\} \in \mathcal{E}(\mathbf{M}(G))$ iff there is an arrow between X and Y or if they are both parents of some common node, for all $X, Y \in \mathcal{V}(G)$. ■

Corollary 2.1. Let B be a Bayesian network and G its underlying graph, then B can be transformed to a Markov network M by

- letting the moral graph of G be the graph of N
- letting $\Phi(M) = \Theta(B)$, i.e. the factors of N are the CPTs of B

The cliques of M then correspond with parents and their children, and the normalisation constant Z is equal to 1, because B already has a proper probability distribution. □

Definition 2.4. Let N be either a Bayesian or a Markov network and let $V \subseteq \mathcal{V}$ be a subset of the variables. We assume that $V = \{X_1, \ldots, X_\ell\}$. An instantation $v \in \prod \Omega(V)$ is called a MAP INSTANTIATION if

$$v \in \arg\max_{x \in \prod \Omega(V)} \mathcal{P}(\forall 1 \leq i \leq \ell \colon X_i = x_i).$$

Finding such a v is called MAP INFERENCE. If $V = \mathcal{V}$, then finding such a v is called MPE INFERENCE. Furthermore, v will be called an MPE INSTANTIATION instead of a MAP instantiation. ■

Example 4. Reusing example 2 for MPE inference, the joint distribution is the product of all the CPTs:

$$\mathcal{P}(A, B, C, D) = \mathcal{P}(A) \cdot \mathcal{P}(B \mid A) \cdot \mathcal{P}(C \mid A) \cdot \mathcal{P}(D \mid B, C)$$

As all the variables are binary, there are only 2^4 combinations to try. A variable assignment with the highest probability (solution to the problem of MPE inference) is given by $\{A = 0, B = 1, C = 0, D = 1\}$, with probability

$$\mathcal{P}(A = 0, B = 1, C = 0, D = 1) = \frac{4}{23}.$$

For MAP inference on the variables $V = \{A, B, C\}$ we first have to marginalise the distribution to the variables in V (i.e. sum out D) and then maximise the resulting distribution. Taking the CPTs out of the sum which do not depend on D yields an easier calculation where we only need to sum out the CPT $\mathcal{P}(D \mid B, C)$. This is also the basis of the variable elimination method in [Dar09].

We have that

$$\sum_{d \in \Omega(D)} \mathcal{P}(A, B, C, D = d) = \mathcal{P}(A) \cdot \mathcal{P}(B \mid A) \cdot \mathcal{P}(C \mid A) \cdot \sum_{d \in \Omega(D)} \mathcal{P}(D = d \mid B, C)$$

where $\sum_{d \in \Omega(D)} \mathcal{P}(D = d \mid B, C)$ is given by

B	C	$\sum_{d \in \Omega(D)} \mathcal{P}(D = d \mid B, C)$
0	0	1
0	1	1
1	0	1
1	1	1

Note that marginalised CPTs do not necessarily represent probability distributions anymore.

Maximising $\sum_{d \in \Omega(D)} \mathcal{P}(A, B, C, D)$ yields the assignment $\{A = 0, B = 1, C = 0\}$ with probability

$$\sum_{d \in \Omega(D)} \mathcal{P}(A = 0, B = 1, C = 1, D = d) = \frac{1}{4}.$$

■

Chapter 3

Exact methods

3.1 Introduction

This chapter will introduce various methods for exact calculation of MPE instantiations. First we will show how the problem of MPE inference in a Markov network (and by extension, in a Bayesian network) can be converted into a equivalent integer linear program (ILP) and we will give the complexity of this procedure. We will also give an example network and its translation into an ILP problem.

At the end of the chapter, there is a small section on incorporating constraints from other networks over the same variables as well as a section on a weighted model counting formulation of MPE inference in Bayesian networks. These subjects will however not be further elaborated on in the rest of the thesis and are meant as resp. an example of a benefit which the ILP formulation provides us and as an example of another exact inference procedure for Bayesian networks.

Definition 3.1. An INTEGER LINEAR PROGRAM is a mathematical procedure which maximises a linear function subject to linear constraints. It consists of a vector of positive integer variables x, a linear objective function over x and linear constraints over x. In canonical form it is expressed as

$$\arg\max_{x \in \mathbf{Z}^m_{\geq 0}} \{\langle c, x \rangle : Ax \leq b\}.$$

In the above formulation, the variables are an n-dimensional $\mathbf{Z}_{\geq 0}$-vector and the objective function is given by $\langle c, x \rangle$ where $c \in \mathbf{R}^n$. Moreover, the constraints are expressed by $Ax \leq b$, for some $A \in \mathrm{Mat}_{m \times n}(\mathbf{R})$ and $b \in \mathbf{R}^m$. Note that the coefficients of the objective function and the constraints are not necessarily integers. ■

3.2 Converting a probabilistic network to an ILP problem

Some of the ideas in this section were inspired by [Nie]. Given a Markov network M we would like to find the most probable assignment to the variables. To this end we will convert M to a canonical ILP as defined in the introductory section. We will first define the necessary variables, then give the constraints and finally find a linear objective function which is equivalent to the joint distribution of M with respect to maximisation. That is to say, a variable assignment is maximal in the joint distribution iff it is maximal in the objective function.

Suppose that M contains the variables $\mathcal{V} = \{X_i\}_{i=1}^n$ where $\Omega(X_i) = \{a_1^i, \ldots, a_{n_i}^i\}$ for all $1 \leq i \leq n$. Furthermore, let $\ell = \#\mathcal{C}$. For all $1 \leq i \leq n$ there exists a mapping $\gamma_i \colon \Omega(X_i) \to \{k \in \mathbf{Z} \colon 1 \leq k \leq n_i\}$ defined by $a_k^i \mapsto k$, preserving any ordering $\Omega(X_i)$ may have. Without loss of generality we will henceforth identify $\Omega(X_i)$ with $\gamma_i(\Omega(X_i))$, i.e. a value will be identified with its index in the ordering.

The factor of the i-th clique is denoted by w_i. The value of the j-th row in w_i, given an ordering of the variables in the i-th clique, is denoted by w_{ij}. The rows are sorted lexographically, starting from the first index. Assume that factor w_i has r_i rows.

row	X_1	X_2	$\varphi(X_1, X_2)$
0	0	0	$\varphi(X_1 = 0, X_2 = 0)$
1	0	1	$\varphi(X_1 = 0, X_2 = 1)$
2	0	2	$\varphi(X_1 = 0, X_2 = 2)$
3	1	0	$\varphi(X_1 = 1, X_2 = 0)$
4	1	1	$\varphi(X_1 = 1, X_2 = 1)$
5	1	2	$\varphi(X_1 = 1, X_2 = 2)$

Table 3.1: A table which shows the ordering of the rows in a factor φ, with $\Omega(X_1) = \{0, 1\}$ and $\Omega(X_2) = \{0, 1, 2\}$.

3.2.1 Variables

There are two kinds of variables, called x and y. They are both binary variables, meaning that they assume values from $\{0, 1\}$.

The variable x relates the network variables to their values. We have that

$$x_{ij} = 1 \iff X_i = j$$

for all $1 \leq i \leq n$ and $j \in \Omega(X_i)$. Note that for all $1 \leq i \leq n$ there exists a unique j such that $x_{ij} = 1$, because a variable can assume only one value at a time.

The variable y connects the factors with their instantiations. We have that

$$y_{ij} = 1 \iff \text{factor } i \text{ is active at instantiation } j.$$

Just like with the x variable, for every $1 \leq i \leq \ell$ there exists a unique j such that $y_{ij} = 1$, because a factor can have only one active instantiation at a time.

3.2.2 Objective function

Let $x \in \prod \Omega(\mathcal{V})$ be a complete assignment, then the probability distribution of M can be rewritten as follows:

$$\begin{aligned}
\mathcal{P}_M(X = x) &\propto \prod_{C \in \mathcal{C}} \varphi_C(X_C = x_C) \\
&= \exp\left(\log\left(\prod_{C \in \mathcal{C}} \varphi_C(X_C = x_C)\right)\right) \\
&= \exp\left(\sum_{C \in \mathcal{C}} \log(\varphi_C(X_C = x_C))\right) \\
&= \exp\left(\sum_{C \in \mathcal{C}} \sum_{u \in \mathrm{dom}(\varphi_C)} \log(\varphi_C(X_C = u)) 1_{X_C = u}(x_c)\right) \\
&= \exp\left(\sum_{C \in \mathcal{C}} \sum_{u \in \prod \Omega(C)} \log(\varphi_C(X_C = u)) 1_{X_C = u}(x_c)\right).
\end{aligned}$$

The function $1_{X_C = u}$ is an indicator function defined by

$$1_{X_C = u}(u') = \begin{cases} 0, & u \neq u' \\ 1, & u = u' \end{cases}$$

for all $u' \in \prod \Omega(C)$.

This representation of the distribution is also known as the *Boltzmann distribution*. It is useful because it is a linear function in the "variables" $1_{X_C = u}$ and coefficients $\log(\varphi_C(X_C = u))$ and the goal is to transform the maximisation of the probability distribution into an ILP problem.

The indicator functions in the derived expression above can be replaced by the y-variables from the previous section. Furthermore, the exponential function on the outside can be dropped because maximising $\exp \circ f$ is equivalent to maximising f for any real-valued function f.

We will therefore optimise

$$z = \sum_{i=1}^{\ell} \sum_{j=1}^{r_i} \log(w_{ij}) y_{ij}$$

given the constraints from the next section.

3.2.3 Constraints

We will require that $\sum_{j=1}^{n_i} x_{ij} = 1$ for all $1 \leq i \leq n$, as a variable X_i can assume only one value j at a time. Furthermore, $y_{ij} = 1$ holds for a unique j for every factor i, while $y_{ij} = 0$ for all other j. This can be accomplished by the equalities $\sum_{j=1}^{r_i} y_{ij} = 1$ for all $1 \leq i \leq \ell$.

Let $\iota \colon \{X_i\}_{i=1}^{n} \to \mathbf{Z}_{\geq 1}$ be defined by $X_i \mapsto i$, then $\iota(C) = \{j \colon X_j \in C\}$ is the set of all indices of clique $C \in \mathcal{C}$. Let v_{ijk} denote the value of variable X_k in row j of factor w_i. Then $y_{ij} = 1$ iff the variables assume the values of the j-th row in CPT w_i, if

$$\sum_{k \in \iota(C_i)} x_{kv} - y_{ij} \leq \#\iota(C_i) - 1 = \#C_i - 1$$

and

$$y_{ij} - x_{kv} \leq 0 \quad \text{for all } k \in \iota(C_i)$$

where $v = v_{ijk}$, $1 \leq i \leq \ell$ and $1 \leq j \leq r_i$.

Example 5. Let X_1 and X_2 be two variables in some factor $\varphi(X_1, X_2)$ and let $y = 1$ be synonymous with $X_1 = 0$ and $X_2 = 1$. If the following constraints hold

$$x_{10} + x_{21} - y \leq 1, \qquad y - x_{10} \leq 0, \qquad y - x_{21} \leq 0,$$

then $y = 1$ implies that $x_{10} = 1$ and $x_{21} = 1$ and $y = 0$ implies that $x_{10} \neq 1$ or $x_{21} \neq 1$. ■

Semantics	Constraints
Domains of variables.	$0 \leq y_{ij} \leq 1$ for all $1 \leq i \leq n, 1 \leq j \leq r_i$ $0 \leq x_{ij} \leq 1$ for all $1 \leq i \leq n, 1 \leq j \leq n_i$
Ensure that every factor can only contribute once to the joint distribution.	$\sum_{j=1}^{r_i} y_{ij} = 1$ for all $1 \leq i \leq \ell$
Ensure that every network variable assumes only one value.	$\sum_{j=1}^{n_i} x_{ij} = 1$ for all $1 \leq i \leq n$
Ensure that y_{ij} represents the correct variable assignment.	$\sum_{k \in \iota(C_i)} x_{kv} - y_{ij} \leq \#C_i - 1$ $y_{ij} - x_{kv} \leq 0$ for all $1 \leq i \leq \ell$, $1 \leq j \leq r_i$ and $v = v(i,j,k)$

Table 3.2: The semantics of each set of constraints.

Example 6. A final example to demonstrate the translation of a Markov network to an ILP problem.

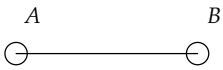


Figure 3.1: A simple Markov network.

The network contains two binary variables, A and B. The cliques are $\mathcal{C} = \{\{A\}, \{B\}, \{A, B\}\}$. The factors are given by

A	$\varphi(A)$
0	$\frac{1}{2}$
1	$\frac{1}{2}$

B	$\varphi(B)$
0	$\frac{1}{3}$
1	$\frac{2}{3}$

A	B	$\varphi(A,B)$
0	0	$\frac{1}{4}$
0	1	$\frac{1}{4}$
1	0	$\frac{1}{4}$
1	1	$\frac{1}{4}$

There are 8 binary y-variables: $Y = \{y_{A0}, y_{A1}, y_{B0}, y_{B1}, y_{AB,0}, y_{AB,1}, y_{AB,2}, y_{AB,3}\}$, one for every row in every factor. Using these y-variables, the objective function z can be written as:

$$z = -y_{A0} \log 2 - y_{A1} \log 2 - y_{B0} \log 3 + y_{B1} \log \frac{2}{3} - y_{AB,0} \log 4 - y_{AB,1} \log 4 - y_{AB,2} \log 4 - y_{AB,3} \log 4.$$

We have rewritten logarithms of the form $\log \frac{1}{x}$ to $-\log x$.

Similar to the y-variables, we have a set of x-variables of cardinality 4: $X = \{x_{A0}, x_{A1}, x_{B0}, x_{B1}\}$, one for every value assignment to every variable. Note that the second indices of the x-variables denote the values of the first indices which are variables, whereas the second index of the y-variables denotes a row number in the associated factor.

This yields the following ILP problem:

$$\max z = -y_{A0}\log 2 - y_{A1}\log 2 - y_{B0}\log 3 + y_{B1}\log\frac{2}{3} - y_{AB,0}\log 4 - y_{AB,1}\log 4 - y_{AB,2}\log 4 - y_{AB,3}\log 4$$

$$\text{subject to} \quad x_{A0} + x_{A1} = 1, \quad x_{B0} + x_{B1} = 1$$

$$y_{A0} + y_{A1} = 1, \quad y_{B0} + y_{B1} = 1, \quad y_{AB,0} + y_{AB,1} + y_{AB,2} + y_{AB,3} = 1,$$

$$x_{A0} - y_{A0} \leq 0, \quad x_{A1} - y_{A1} \leq 0, \quad x_{B0} - y_{B0} \leq 0, \quad x_{B1} - y_{B1} \leq 0,$$
$$x_{A0} + x_{B0} - y_{AB,0} \leq 0, \quad x_{A0} + x_{B1} - y_{AB,1} \leq 0,$$
$$x_{A1} + x_{B0} - y_{AB,2} \leq 0, \quad x_{A1} + x_{B1} - y_{AB,3} \leq 0$$

$$y_{A0} - x_{A0} \leq 0, \quad y_{A1} - x_{A1} \leq 0, \quad y_{B0} - x_{B0} \leq 0, \quad y_{B1} - x_{B1} \leq 0,$$
$$y_{AB,0} - x_{A0} \leq 0, \quad y_{AB,0} - x_{B0} \leq 0, \quad y_{AB,1} - x_{A0} \leq 0, \quad y_{AB,1} - x_{B1} \leq 0,$$
$$y_{AB,2} - x_{A1} \leq 0, \quad y_{AB,2} - x_{B0} \leq 0, \quad y_{AB,3} - x_{A1} \leq 0, \quad y_{AB,3} - x_{B1} \leq 0$$

$$x, y \in \{0,1\} \quad \text{for all ILP variables } x \in X \text{ and } y \in Y$$

■

3.2.4 Incorporating constraints from other networks

Let $\{M_k\}_{k=1}^m$ be Markov networks over the same variables as M and $\theta \in [0,1]^m$, then we would like to calculate $\max_x \mathcal{P}_M(X = x)$ subject to $\mathcal{P}_{M_k}(X = x) \leq \theta_i$ for all $1 \leq k \leq m$. The aforementioned constrained maximisation problem can be solved by extending the ILP problem which we have constructed with the constraints

$$\sum_{i=1}^{n}\sum_{j=1}^{r_i} \log(w_{ij}^k) y_{ij}^k \leq \log(\theta_k) \quad \text{for all } 1 \leq k \leq m,$$

where w_{ij}^k denotes the j-th row in the factor $\varphi_{C_i}^k$ and where y_{ij}^k is a binary variable similar to $y_{ij} = y_{ij}^0$ but belonging to network M_k.

3.2.5 Complexity

For every variable X, we must create $\#\Omega(X)$ ILP x-variables, so $\sum_{X\in\mathcal{V}} \#\Omega(X)$ x-variables need to be created in total. We further need to create $\sum_{C\in\mathcal{C}} \prod_{X\in C} \#\Omega(X)$ ILP y-variables in total.

Creating the objective function takes as many steps as their are y-variable declarations.

Then we must create $\#\mathcal{C}$ constraints of the form $\sum_{j \in \prod \Omega(C)} y_{ij} = 1$ for all cliques $i \in \mathcal{C}$ and $\#\mathcal{V}$ constraints of the form $\sum_{j \in \Omega(X)} x_{ij} = 1$ for all variables $X \in \mathcal{V}$. Creating these constraints costs as many steps in total as creating the variables, because each variable only appears in one unique constraint.

Lastly, the constraints which make sure that the ILP variables represent correct assignments require $\sum_{C \in \mathcal{C}} (\#C + 1) \prod_{X \in C} \Omega(X)$ and $\sum_{C \in \mathcal{C}} 2 \prod_{X \in C} \Omega(X)$ steps. The first number of steps corresponds to constraints of the form $\sum_{k \in \iota(C_i)} x_{kv} - y_{ij} \leq \#C_i - 1$ and the second number of steps is related to the constraints of the form $y_{ij} - x_{kv} \leq 0$. Together this yields $\sum_{C \in \mathcal{C}} (\#C + 3) \prod_{X \in C} \Omega(X)$ steps in total.

Using the reasoning above, we arrive at a total complexity of

$$\begin{aligned} &\mathcal{O}\left(2 \sum_{X \in \mathcal{V}} \#\Omega(X) + 3 \sum_{C \in \mathcal{C}} \prod_{X \in C} \#\Omega(X) + \sum_{C \in \mathcal{C}} (\#C + 3) \prod_{X \in C} \Omega(X)\right) \\ &= \mathcal{O}\left(2 \sum_{X \in \mathcal{V}} \#\Omega(X) + \sum_{C \in \mathcal{C}} (\#C + 6) \prod_{X \in C} \Omega(X)\right). \end{aligned}$$

If we assume that C is the maximum clique size and that ℓ is the maximum number of values any variable can assume, then the complexity derived above is bounded by the following complexity:

$$\mathcal{O}(\#\mathcal{V} \cdot \ell + \#\mathcal{C} \cdot \ell^C).$$

Hence the translation procedure has a complexity that is exponential in the maximum clique size. Still, the steps taken by the algorithm are typically very small, so this conversion can still be done quickly.

3.3 Weighted MAX-SAT

This section gives an example of another exact method for MPE inference in Bayesian networks (not Markov networks). We will provide the details of the weighter MAX-SAT encoding of MPE inference in this section, but will not use it further in this thesis.

The problem of MPE inference in Bayesian networks can be encoded as a weighted MAX-SAT problem as follows (from [Dar09]).

For each variable $X \in \mathcal{V}$ and value $x \in \Omega(X)$ there exists an indicator variable I_x. If the values of X are numbered $x_1, \ldots, x_n$, then the encoding contains the following indicator clauses,

$$\left(\bigvee_{k=1}^{n} I_{x_k}\right)^W \quad \text{and} \quad (\neg I_{x_k} \vee \neg I_{x_l})^W, \quad \text{for all } k < l.$$

The superscript W signifies the weight assigned to that clause. In this case, W is a special weight given to a clause if it needs to be satisfied by any maximal truth assignment. Clauses given this weight are called *hard clauses*, while clauses which do not are called *soft clauses*. The weight W is defined to be at least the sum of weights assigned to all soft clauses. It follows that any optimal truth assignment should satisfy all hard clauses.

Additionally, the weighted MAX-SAT encoding contains parameter clauses for each instantation of every CPT $\theta_{X|\mathrm{par}(X)} \in \Theta$ with m parents,

$$\left(\neg I_x \vee \bigvee_{k=1}^{m} \neg I_{p_k}\right)^{-\log'(\theta_{x|p})}, \quad \text{for all } x \in \Omega(X) \text{ and } p \in \prod \Omega(\mathrm{par}(X)),$$

where $\log'(0) = W$ and $\log'(x) = \log(x)$ for all x in the domain of log.

A truth assignment with minimal penalty now corresponds to an instantation with maximal probability. The penalty of a truth assignment Γ is defined to the sum of all weights of the clauses which are not satisfied by Γ.

Chapter 4

Approximation methods

This chapter will introduce approximation methods for MPE inference. These algorithms can speed up the calculations of exact MPE solutions with the ILP method, by adding the approximation as a lower bound to the ILP problem. These approximations are always a lower bound to the real MPE solution, as every MPE solution is at least as high as any other assignment.

Specifically, we will discuss two algorithms:

- An ant colony optimisation algorithm tailored to search for high probability variable assignments in Bayesian networks. It is based on the ANT-MPE algorithm by [GBH04], but uses a new approach which is meant to give it a clearer explanation and easier programmability.

- An adjusted version of the well-known belief propagation algorithm for Markov networks which is explained in section 4.1. The regular belief propagation algorithm produces exact solutions for tree-shaped networks. Our algorithm is a generalisation to arbitrarily shaped networks. Consequently, our algorithm doesn't necessarily find an exact solution in every network.

  The algorithm we have devised is called BP-ACO, which stands for belief propagation-ant colony optimisation. The latter segment of this name is owed to the part of the algorithm which uses ant colony optimisation to search for "good" spanning trees of the network.

In the rest of this chapter, M shall be a Markov network. Remember that there is a bijective correspondence between the cliques $\mathcal{C}$ and the factors Φ: for every clique $C \in \mathcal{C}$ there is a factor $\Phi \ni \varphi_C \colon \prod \Omega(C) \to \mathbf{R}_{\geq 0}$ over the variables in C.

4.1 Belief propagation for MPE inference

Belief propagation was originally created by Judea Pearl in 1985 in his paper [Pea85] as a method for performing inference in tree-shaped Bayesian networks. It is a remarkable algorithm because it reduces the problem of inference to a set of smaller problems which use local neighbour information to be computed. Later, the technique of belief propagation was generalised to arbitrary Bayesian networks (not necessarily trees) and Markov networks (see, for example, [YFW01]).

4.1.1 Definition

This definition of the belief propagation algorithm for MPE inference was inspired by [YFW01]. In order to talk about belief propagation for tree-shaped networks we will first need the following definition.

Definition 4.1. The FACTOR GRAPH $F(M)$ of the Markov network M is a bipartite graph with variable nodes $\mathcal{V}$ and factor nodes $\mathcal{C}$. There is an edge between a variable node $v \in \mathcal{V}$ and a factor node $C \in \mathcal{C}$ if and only if $v \in C$. ■

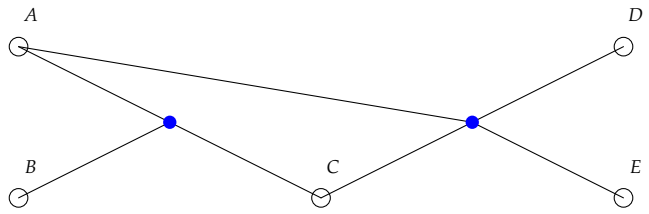


Figure 4.1: An example factor graph. The blue vertices are clique nodes; the white vertices are variable nodes. In this case there are two cliques: $\{A, B, C\}$ (left-hand blue dot) and $\{A, C, D, E\}$ (right-hand blue dot).

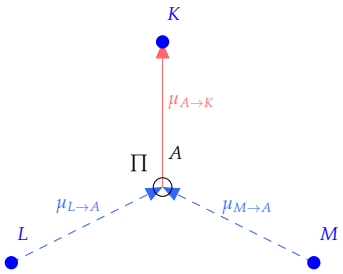
For each edge $(v, C) \in \mathcal{V} \times \mathcal{C}$ in the factor graph $F(M)$ there exist two real-valued message functions: $\mu_{v\to C}$ and $\mu_{C\to v}$. These message functions are defined as

$$\mu_{v\to C}(x) = \prod_{\substack{C^* \in \mathcal{C} \\ C^* \neq C,\, v \in C^*}} \mu_{C^*\to v}(x), \quad \forall x \in \Omega(v) \tag{4.1}$$

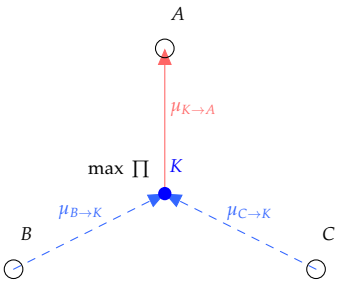
and

$$\mu_{C\to v}(x) = \max_{\substack{y \in \prod \Omega(C) \\ y_v = x}} \varphi_C(y) \prod_{\substack{v^* \in V \\ v^* \neq v,\, v^* \in C}} \mu_{v^*\to C}(y_{v^*}), \quad \forall x \in \Omega(v)\,. \tag{4.2}$$

Here, $y_v = x$ means that the v-component of y is equal to x. When v has no neighbouring factor nodes other than C, $\mu_{v\to C}(x)$ is set to a uniform distribution, i.e. all assignments have equal probability.



(a) A message from variable A to clique K



(b) A message from clique K to variable A

Figure 4.2: A display of the two kinds of message propagation. The network contains the variables A, B and C and the cliques K, L and M. In Figure (a), $\mu_{A\to K}$ is updated as $\mu_{A\to K}(a) := \mu_{L\to A}(a) \cdot \mu_{M\to A}(a)$ for all $a \in \Omega(A)$, using equation 4.1. In Figure (b), $\mu_{K\to A}$ is updated as $\mu_{K\to A}(a) := \max_{x\in\Omega(A)\times\Omega(B)\times\Omega(C)\wedge x_A=a} \varphi_K(x) \cdot \mu_{B\to K}(x_B) \cdot \mu_{C\to K}(x_C)$ for all $a \in \Omega(A)$, using equation 4.2.

Messages are similar to CPTs, but they do not necessarily have to obey all the probability laws; messages are not probability distributions in general.

4.1.2 How the algorithm works

The algorithm begins by choosing a root node of the tree-shaped network. Then, starting from the leaves, messages are updated inwards in the direction of the root in the *pull* phase. The messages only have to be computed once to arrive at an exact solution. Once all the messages have been propagated to the root, the MPE assignment is reconstructed from the messages in the *push* phase. These statements shall be made more precise in Section 4.3 where we discuss a generalisation of this algorithm.

4.2 Ant colony optimisation

Real-world ants have the ability to cooperate for certain tasks and find short paths in a probabilistic manner using pheromone[1] trails. The more ants follow a certain path, the more attractive that path becomes resulting in more and more ants following it.

This section is based on [GBH04].

4.2.1 What is ant colony optimisation formally?

Ant colony is a probabilistic optimisation technique used to calculate approximations of optimal paths in graphs.

[1] Pheromones are excreted chemicals used by animals as a form of communication.

It requires the specification of three things:

(1) The graph which needs to be searched, with costs on its edges.

(2) A pheromone table (PT) τ, which contains numbers signifying the attractiveness for every edge in the graph. Every time a path is found, the pheromones are updated with respect to the costs of this path. The PT functions as a communication device between the ants and is initialised uniformly at the start of the algorithm.

(3) A heuristic table (HT) η, which also contains numbers that represent desirability for every edge in the graph. The difference between the HT and the PT is that the former is a static matrix (i.e. it isn't updated by the algorithm), while the latter is updated after every iteration.

The PT and HT are used to calculate transition probabilities for edges in the graph. Higher pheromone and heuristic values for some edge therefore translate to a higher probability that ants will follow that edge.

We will use ant colony optimisation to find good solutions to the MPE problem in Bayesian networks. In order to do this, we will not traverse the Bayesian network directly. Instead, we will transform the network into a Markov model[2] which captures all possible assignments to the network variables. For visualisation we shall use the so-called *assignment tree*, which will be explained in the next section. The assignment tree is not explicitly constructed as it is a search tree; only the branches which are used by an ant are in reality evaluated. This Markov model depends on a particular topological ordering[3] of the network variables; any will do. This remark shall be further explained below. The algorithm that we will discuss now is based on the ANT-MPE algorithm in [GBH04], but uses a different approach.

4.2.2 Applying ant colony optimisation to MPE inference

Let $(X_1, X_2, \ldots, X_m)$ be such a topological ordering. Each node in the assignment tree denotes a particular value of a network variable, along with the values of its parents. Level i of the tree corresponds to variable X_i in the topological ordering and the root node has level 0. Moreover, every node on level i ($i < m$) has outgoing edges for every value in $\Omega(X_{i+1})$ weighted by the transition probabilities defined later.

[2] A *Markov model* is (probabilistic) model of random variables which assumes the Markov property. This means that the value of variable X_{i+1} depends only on the value of X_i in a stochastic manner.

[3] A *topological ordering* of the partially ordered network variables is an ordering such that every variable appears after its parents. As a consequence, any variable without parents can appear as the first element in this ordering.

Example 7. We have a network with a ternary variable A and a binary variable B with values $\Omega(A) = \{0, 1, 2\}$ and $\Omega(B) = \{0, 1\}$. The network structure is given by

Figure 4.3: A Bayesian network for example 7.

with the CPTs

$\mathcal{P}(A=0)$	$\mathcal{P}(A=1)$	$\mathcal{P}(A=2)$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{2}{3}$

and

A	$\mathcal{P}(B=0)$	$\mathcal{P}(B=1)$
0	$\frac{1}{2}$	$\frac{1}{2}$
1	$\frac{2}{3}$	$\frac{1}{3}$
2	$\frac{5}{7}$	$\frac{2}{7}$

The assignment tree is displayed in figure 4.5 using the topological ordering (A, B).

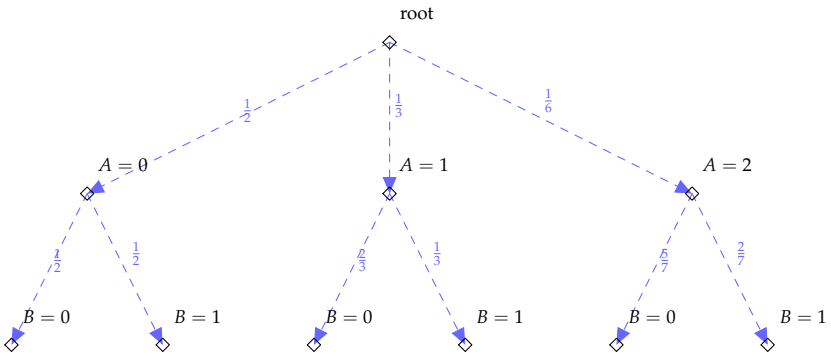


Figure 4.5: The assignment tree. Note that variables which are connected in this tree are not necessarily connected in the Bayesian network. A simple example of this is the network with the variables A, B and C with C having A and B as parents; A and B have no parents themselves. This network can be sorted topologically either as (A, B, C) or as (B, A, C). The variables A and B are thus connected in the assignment tree, but not in the network.

There are two optimal paths in this tree with value $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, corresponding to two MPE assignments: $(A = 0, B = 0)$ and $(A = 0, B = 1)$. Note that there can be multiple nodes which represent the same value assignment to some variable, for example there are tree $(B = 0)$-nodes. This is true in general for any variable of level i with $i > 1$. ■

The PT is defined up to value equivalence of edges. This means that if we have two different edges $e_1 = (B = 0, C = 1)$ and $e_2 = (B = 0, C = 1)$ (their parents have had different values assigned to them), they both share the same pheromone $\tau_{B=0,C=1}$. If every unique edge were to have its own pheromones, then no information is shared between different assignments. In our construction, pheromones let an assignment increase the attractiveness of any other assignment if they agree on some consecutive variables in the topological ordering.

The HT is in 1-to-1 correspondence with the network's CPTs and thus contains an entry for each unique edge. This is because a CPT can assume different values depending on the parents' values. For example, if A, B and C are three binary variables and are associated to a CPT $\mathcal{P}(C \mid A, B)$, then we do not necessarily have that $\mathcal{P}(C = 0 \mid A = 0, B = 0) = \mathcal{P}(C = 0 \mid A = 0, B = 1)$.

To simplify the formulas, every node in the assignment tree will additionally be labelled with its parents' assignments. Let X_i and X_{i+1} be two variables, $x_i \in \Omega(X_i)$ and $x_{i+1} \in \Omega(X_{i+1})$, and A the assignments to their parents. Then the transition probability from $s \equiv_\alpha (X_i = x_i)$ to $t \equiv_\alpha (X_{i+1} = x_{i+1})$ given assignment A is given by

$$p_{st}(A) \propto (\tau_{st})^\alpha (\eta_{st}(A))^\beta$$

i.e.

$$p_{st}(A) = \frac{(\tau_{st})^\alpha (\eta_{st}(A))^\beta}{\sum_{\substack{x'_{i+1} \in \Omega(X_{i+1}) \\ t' \equiv_\alpha (X_{i+1} = x'_{i+1})}} (\tau_{st'})^\alpha (\eta_{st'}(A))^\beta} = \frac{(\tau_{st})^\alpha (\mathcal{P}(X_{i+1} = x_{i+1} \mid X_i = x_i, A))^\beta}{\sum_{\substack{x'_{i+1} \in \Omega(X_{i+1}) \\ t' \equiv_\alpha (X_{i+1} = x'_{i+1})}} (\tau_{st'})^\alpha \left(\mathcal{P}\left(X_{i+1} = x'_{i+1} \mid X_i = x_i, A\right)\right)^\beta}, \quad (4.3)$$

where $\alpha, \beta \in \mathbf{R}$ are parameters of the algorithm. Other parameters are the number of ants that traverse the graph each iteration and either the number of iterations the algorithm has to run or a halting predicate which judges the current best solution, possibly taking into account previous solutions and other factors.

If we view the topological ordering of variables as a sequence of states, the assignment tree and its transition probability formulas actually give rise to a Markov model where the value of state $k + 1$ depends only on the value of state k.

Now assume that there are a ants. Then ant k which has found the complete assignment A after having traversed the graph leaves $\Delta\tau^k$ pheromones on each edge, defined by

$$\Delta\tau^k_{st} = \begin{cases} \mathcal{P}(A), & \text{if } s \text{ and } t \text{ are in ant } k\text{'s path and } t \text{ follows } s \text{ in the topological ordering} \\ 0, & \text{otherwise} \end{cases}$$

The variables s and t have a similar meaning to the s and t above in the transition probability formula.

The pheromone table is updated as follows:

$$\tau_{st} := (1 - \rho)\tau_{st} + \Delta\tau_{st}, \quad (4.4)$$

where $\Delta\tau_{st} = \sum_{k=1}^{a} \Delta\tau^k_{st}$ and $\rho \in [0, 1]$ is a constant which controls the rate at which old pheromones dissipate.

Data: B: Bayesian network.
Result: A complete assignment of the variables in the Bayesian network B.
$\alpha, \beta \in \mathbf{R}$ (parameters)
$m \in \mathbf{Z}_{>0}$ (number of ants)
$t = 1$ (iteration counter)
$(X_1, \ldots, X_{\#\mathcal{V}}) :=$ topologically sorted array of the variables in the network
$T :=$ assignment tree using the topologically sorted array
$\tau :=$ uniform initialisation of the pheromones
$B :=$ randomly initialised best assignment
while *no satisfactory assignment has been found* **do**
  $p :=$ calculate the transition probabilities using equation 4.3
  foreach *ant i from* 1 *to m* **do**
    $A :=$ assignment resulting from traversal of the assignment tree with probabilities p
    if $\mathcal{P}(B) < \mathcal{P}(A)$ **then**
      $B := A$ (the best assignment B is not as good as the current assignment A)
    end
  end
  $\tau := (1-\rho)\tau + \sum_{i=1}^{m} \Delta\tau_i$ (equation 4.4)
  $t := t + 1$
end

Algorithm 1: The ant colony algorithm.

Example 8. This example reuses the network from example 7 to walk through the ant colony algorithm. We will use one ant per iteration, $\alpha = \beta = 1$, $\rho = 0$ and as many iterations as necessary (in this case 2). First we sort the variables topologically: (A, B) is the only way this can be done. Given this ordering of the variable, we get the same assignment tree as displayed in figure 4.5. There are 9 different branches up to value equivalence, so to initialise the pheromones, every branch gets a pheromone value of $\frac{1}{9}$. We take $\{A = 2, B = 1\}$ to be our initial best assignment, with value $\frac{1}{6} \cdot \frac{2}{7} = \frac{1}{21}$.

Iteration 0

The ant is now located at the root node. The transition probabilities are given by the CPT/edge probabilities, because the pheromones are uniformly initialised, so

$$p_{\text{root},A=0} = \mathcal{P}(A = 0) = \frac{1}{2}, \qquad p_{\text{root},A=1} = \mathcal{P}(A = 1) = \frac{1}{3}, \qquad p_{\text{root},A=2} = \mathcal{P}(A = 2) = \frac{1}{6}.$$

By chance, we choose $A = 1$. We can now choose beteen $B = 0$ with probability $\frac{2}{3}$ and $B = 1$ with probability $\frac{1}{3}$, because we are at the node $A = 1$. By chance, we choose $B = 0$. The result of this iteration is the assignment $\{A = 1, B = 0\}$ with value $\frac{2}{9}$. As for the pheromone update: all edges keep their current pheromones, except for the edges in this iteration's ant's path. We therefore have $\tau_{\text{root},A=1} := \frac{1}{9} + \frac{2}{9} = \frac{1}{3}$ and $\tau_{A=1,B=0} := \frac{1}{9} + \frac{2}{9} = \frac{1}{3}$.

The path that was traversed by the ant is indicated by the red dots in figure 4.6.

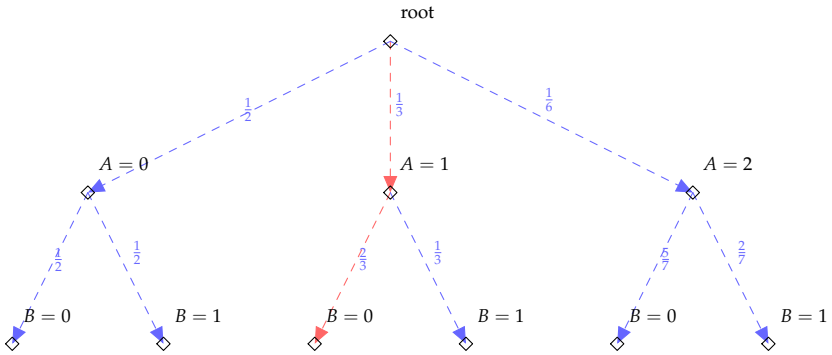


Figure 4.6: The assignment tree after it has been crossed by the ant in iteration 0.

Iteration 1

The ant is located at the root node again. The transition probabilities are now:

$$p_{\text{root},A=0} = \frac{\frac{1}{2}\cdot\frac{1}{9}}{\frac{1}{2}\cdot\frac{1}{9}+\frac{1}{3}\cdot\frac{1}{3}+\frac{1}{6}\cdot\frac{1}{9}} = \frac{\frac{1}{18}}{\frac{5}{27}} = \frac{3}{10},$$
$$p_{\text{root},A=1} = \frac{\frac{1}{3}\cdot\frac{1}{3}}{\frac{1}{2}\cdot\frac{1}{9}+\frac{1}{3}\cdot\frac{1}{3}+\frac{1}{6}\cdot\frac{1}{9}} = \frac{\frac{1}{9}}{\frac{5}{27}} = \frac{3}{5},$$
$$p_{\text{root},A=2} = \frac{\frac{1}{6}\cdot\frac{1}{9}}{\frac{1}{2}\cdot\frac{1}{9}+\frac{1}{3}\cdot\frac{1}{3}+\frac{1}{6}\cdot\frac{1}{9}} = \frac{\frac{1}{54}}{\frac{5}{27}} = \frac{1}{10}.$$

This time, we now choose $A = 0$ by chance. Because the pheromones did not change for this branch, the transition probabilities at node $A = 0$ are equal to the CPT probabilities: $p_{A=0,B=0} = \mathcal{P}(B = 0 \mid A = 0) = \frac{1}{2}$ and $p_{A=0,B=1} = \mathcal{P}(B = 1 \mid A = 0) = \frac{1}{2}$. By chance, we choose $B = 0$ and end up with the complete assignment $\{A = 0, B = 0\}$ with value $\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}$. The new pheromones are $\tau_{\text{root},A=0} := \frac{1}{9}+\frac{1}{4} = \frac{13}{36}$ and $\tau_{A=0,B=0} := \frac{1}{9}+\frac{1}{4} = \frac{13}{36}$; the other pheromones stay the same as in the previous iteration.

The path that was traversed by the ant is indicated by the red dots in Figure 4.7.

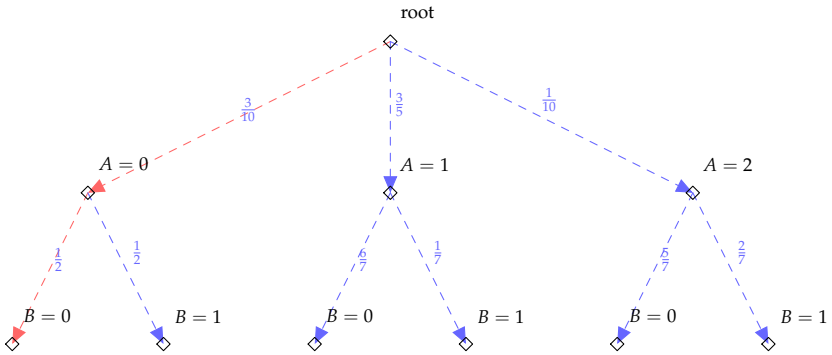


Figure 4.7: The assignment tree after it has been crossed by the ant in iteration 1, with updated transition probabilities. Note that the algorithm would not really calculate the transition probabilities of all the edges in the assignment tree as this would result in an exponential blow-up that would defeat the algorithm's purpose.

This assignment is better than the previous assignment because $\frac{2}{9} < \frac{1}{4}$ and is in fact an MPE assignment, so we are in luck. The algorithm ends with the best MPE assignment $\{A = 0, B = 0\}$. ■

4.3 Generalising belief propagation

As mentioned earlier, the belief propagation procedure given above cannot be used for networks without a tree structure. In this case, a spanning tree of the factor graph is chosen along which messages are propagated using a push-and-pull schema ([Daro9]), in the generalised belief propagation algorithm from now on referred to as BP-ACO. After the messages have been calculated in the spanning tree, the solution is reconstructed and used to update the pheromones for the next iteration. The algorithm halts after it has run for the specified number of iterations or if the solution is deemed to be good enough.

4.3.1 Combining belief propagation and ant colony optimisation

The core of BP-ACO consists of BP, enhanced by a spanning tree search procedure using ACO. At the beginning of the algorithm, the messages are initialised uniformly, as shown in algorithm 2. In every iteration, an arbitrary spanning tree of the factor graph is chosen for every ant using the SpanningTree algorithm with neighbour transition probabilities depending on the pheromones and the messages. Every edge in the tree is an edge between a variable and a clique by the design of the factor graph. Then the messages of the network are propagated in the spanning tree from the leaves to the root in the belief propagation phase. Finally, a variable assignment $A \in \prod \Omega(\mathcal{V})$ is collected which may or may not be an improvement to the currently best assignment.

Data: M: Markov network.
Result: Initialised messages.
foreach $v \in \mathcal{V}$ **do**
  foreach $C \in \mathcal{C}$ *such that* $v \in C$ **do**
    foreach $x \in \Omega(v)$ **do**
      $\mu^1_{v\to C}(x) := \frac{1}{\#\Omega(v)}$ (uniform initialisation)
      $\mu^1_{C\to v}(x) := \max_{\substack{y\in\prod\Omega(\ell)\\ y_v=x}} \varphi_C(y)$
    end
  end
end
return μ

Algorithm 2: This algorithm initialises messages for use in algorithm 6.

Ant colony equations for the spanning tree algorithm

Let $\{v, C\}$ be an edge with $v \in \mathcal{V}$ and $C \in \mathcal{C}$, then

$$\Delta\tau_{vC} = \Delta\tau_{Cv} = \begin{cases} \mathcal{P}(A), & \{v, C\} \text{ belongs to the spanning tree} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

The pheromone update rule is equivalent to the one given in Section 4.2.

The starting node n of the spanning tree may satisfy

$$n \in \arg\max_{i \in \mathcal{V} \cup \mathcal{C}} \sum_{j \in N(i)} \tau_{ji} \tag{4.6}$$

(i.e. a node with the most pheromones on its neighbour edges) and neighbour transition probabilities p given by

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{j \in N(i)} \tau_{ij}^{\alpha} \eta_{ij}^{\beta}},$$

where $N(i) = \{j \in \mathcal{V} \cup \mathcal{C} \colon \{i, j\} \in \mathcal{E}(F(M))\}$ is the set of neighbours of i. Recall that $\mathcal{E}(F(M))$ are the edges of the factor graph of the Markov network M.

The symbol η_{ij} is the heuristic attractiveness of the edge (i, j); it is proportional to some function of the message passed from i to j. We could for example take the average of a message ($\mu \mapsto \frac{1}{n} \sum_x \mu(x)$ where x ranges over the n possible inputs) or the maximal value of a message ($\mu \mapsto \max_x \mu(x)$). In algorithm 6, this function is indicated by f: the *heuristic value estimation function*. Not that $\eta_{ij} \neq \eta_{ji}$ in general, while it is true that $\tau_{ij} = \tau_{ji}$.

Spanning tree

The process of creating a spanning tree in a graph with loops will always cause at least one connection between a clique and a variable node to be "lost". In such a situation, the algorithm will create dummy variable nodes for every clique node which has no connections with a subset of its variables. These dummy variables assume the same values as the real variables. In the variable assignment reconstruction phase of the algorithm, the values of the dummy variables are discarded while the values of real variables they represent are kept.

The SpanningTree algorithm (algorithm 3) creates a spanning tree by repeatedly adding an edge from the current node to an unvisited node to the tree with a certain transition probability. If the current node has no unvisited neighbours and the spanning tree does not have $\#nodes - 1$ edges, then we backtrack to previously visited nodes until we reach a node with unvisited neighbours.

Data: $G = (V, E)$: Graph, p: Matrix of neighbour transition probabilities, n: Starting node.
Result: A spanning tree of G.
$V' := \{i\}$ (Used to keep track of the visited nodes)
$T := \varnothing$ (Spanning tree)
$i := n$ (Current node)
$s := (i)$ (Stack used for backtracking)
while $V' \neq V$ **do**
  while *i has no unvisited neighbours* **do**
    pop previous node l from the stack s
    $i := l$
  end
  $i := j$ with probability p_{ij}
  push j onto the stack s
  $V' := V' \cup \{j\}$
  $T := T \cup \{\{i, j\}\}$
end
return T

Algorithm 3: The SpanningTree algorithm. Given a graph G, a distinguished node n in that graph and neighbour transition probabilities for every node, it finds a spanning tree of G starting from node n according to the transition probabilities.

Belief propagation

In the belief propagation phase of the algorithm, messages (which represent the belief) are propagated from the leaves to the root node of the spanning tree found by algorithm 3. The messages are updated using equations 4.1 (if the leaf node is a variable node) and 4.2 (if the leaf node is a clique node).

Data: M: Markov network, μ: Messages of M, T: Spanning tree of underlying graph of M, n: Root node of T, A: Assignment table.
Result: Updated messages μ and assignment table A.
while $T \neq \{n\}$ **do**
  let ℓ be a leaf node $\neq n$ of T with neighbour j
  $T := T \setminus \{\{\ell, j\}\}$
  if *ℓ is a variable node* **then**
    foreach $x \in \Omega(\ell)$ **do**
      $\mu^{t+1}_{\ell \to j}(x) := \prod_{\substack{C^* \in \mathcal{C} \\ C^* \neq j,\, \ell \in C^*}} \mu^t_{C^* \to \ell}(x)$ (equation 4.1)
    end
  else
    foreach $x \in \Omega(j)$ **do**
      let $g\colon \prod \Omega(\ell) \to \mathbf{R}$ be defined by $y \mapsto \varphi_\ell(y) \prod_{\substack{v^* \in \mathcal{V} \\ v^* \neq j,\, v^* \in \ell}} \mu^t_{v^* \to \ell}(y_{v^*})$
      $A^{t+1}_{\ell \to j}(x) := \arg\max_{\substack{y \in \prod \Omega(\ell) \\ y_j = x}} g(y)$
      $\mu^{t+1}_{\ell \to j}(x) := \max_{\substack{y \in \prod \Omega(\ell) \\ y_j \sim x}} g(y)$ (equation 4.2)
    end
  end
end

Algorithm 4: The BeliefPropagation algorithm for use in algorithm 6. Note that if ℓ is not a variable node (i.e. a clique node), $\mu^{t+1}_{\ell \to j}(x_j)$ can be immediately computed using $A^{t+1}_{\ell \to j}(x_j)$.

Assignment retrieval

In algorithm 6, $V = \mathcal{V} \cup \mathcal{C}$ are the vertices of the factor graph. Furthermore, an assignment table A is used for each message going from a clique to a variable. Given a variable in a clique, for each value it can assume, an assignment of values to the other variables in the clique is stored which maximises the message from the clique to variable. When the root node of the spanning tree is reached, the complete assignment can be reconstructed recursively from A, with a dynamic programming-like procedure.[4]

If the root node is a variable node X, then we maximise the product of its incoming messages, resulting in an optimal value x. The maximal assignments with respect to $X = x$ are read from the assignment tables $A_{C \to X}(x)$ belonging to each child C of X. Because of the tree structure, there are no assignment conflicts if there are multiple children, because the only variable the child cliques have in common is the root node.

If the root node is a clique node C, then the procedure from the preceding paragraph is executed on all C's children, collecting all the assignments of the subtrees and merging them.

Starting from the root node, walking down the branches of the spanning tree, there are two cases: either the current node is a branching node or it is a leaf.

- In a leaf node, nothing needs to be done.
- In a branching node, there are again two cases which need to be distinguished: the node is either a clique node or a variable node. If the focused node is a clique node C with parent node X which has x as its optimal value, then the assignment values of the children of C are given by $A_{C \to X}(x)$. We continue repeating this process, culling new variable assignments and extending the assignment, until the leaf nodes are reached. Note that the variable nodes are skipped, but remembered as parents.

This entire process can be viewed in figure 4.8.

[4] *Dynamic programming* is a technique used to efficiently compute answers to problems with many overlapping subproblems. A simple example is computing the n-th Fibonacci number F_n, defined by the recurrence relation

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n \geq 2 \end{cases}.$$

Computing F_4 gives $F_4 = F_2 + F_3 = (F_0 + F_1) + (F_1 + F_2) = (0 + 1) + (1 + (0 + 1)) = 3$. In this example the term F_2 is computed twice, which isn't very hard to calculate, but the amount of evaluations of a term increases exponentially with n.

Dynamic programming solves this complexity issue by storing every term in memory such that it only has to be computed once. For successive uses of some term, it can be read from memory again instead of causing a waterfall of evaluations. Of course, this comes at the cost of increased memory usage.

Data: M: Markov network, μ: Messages of M, T: Spanning tree of underlying graph of M, n: Root node of T, A: Assignment table.
Result: A complete assignment to all the variables in the network.
if *n is a clique node* **then**
  return $\bigcup_{\text{children } c \text{ of } n}$ RetrieveAssignment(M, μ, T, c, A) (take the union of the assignments of the subtrees)
else if *n is a variable node* **then**
  $x :=$ element of $\arg\max_{x_n \in \Omega(n)} \prod_{\substack{C \in \mathcal{C} \\ n \in C}} \mu_{C \to n}(x_n)$
  $\mathbf{Q} := \{(n, x)\}$ (assignment set holding pairs of (variable, value))
  while *T is being traversed from its root node n to its leaf nodes with c being the focused node* **do**
    let $\pi(c)$ be c's parent in T
    let v be $\pi(c)$'s value in $\mathbf{Q}$, such that $(\pi(c), v) \in \mathbf{Q}$
    $\mathbf{Q} := \mathbf{Q} \cup A_{c \to \pi(c)}(v)$
  end
  return $\mathbf{Q}$
end

Algorithm 5: The RetrieveAssignment algorithm which deduces a (locally) optimal assignment to all the variables for use in algorithm 6. The manner in which the tree T is traversed is implementation-dependent. Both DFS and BFS could be valid choices for tree traversal.

Data: M: Markov network, f: Heuristic value estimation function.
Result: An MPE approximation of M as a variable assignment.
$\alpha, \beta \in \mathbf{R}$ (parameters)
$m \in \mathbf{Z}_{>0}$ (number of ants)
$t := 1$ (message iteration counter)
$T := \varnothing$ (spanning tree of the network)
$\tau := (\tau_{ij})_{i,j=1}^{\#V}$ with $\tau_{ij} = \begin{cases} 0, & \{i,j\} \in \mathcal{E} \\ \frac{1}{\#V}, & \{i,j\} \notin \mathcal{E} \end{cases}$ (pheromone matrix)
$A =$ assignment table
$B =$ randomly initialised best assignment
initialise messages μ using algorithm 2
while *no satisfactory assignment has been found* **do**
  foreach *ant i from 1 to m* **do**
    $\eta := (\eta_{ij})_{i,j=1}^{\#V}$ with $\eta_{ij} = f(\mu_{i \to j})$ (modify the heuristic values according to updated messages)
    $p := (p_{ij})_{i,j=1}^{\#V}$ with $p_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{j \in N(i)} \tau_{ij}^{\alpha} \eta_{ij}^{\beta}}$
    $T :=$ SpanningTree(M, p, n) with n the root node of the tree satisfying equation 4.6
    BeliefPropagation(M, μ^t, T, n, A) (algorithm 4)
    $\mathbf{Q}_i :=$ RetrieveAssignment(M, μ^t, T, n, A) (algorithm 5)
    if $\mathcal{P}(B) < \mathcal{P}(\mathbf{Q}_i)$ **then**
      $B := \mathbf{Q}_i$ (the best assignment B is not as good as the current assignment $\mathbf{Q}_i$)
    end
  end
  $\tau := (1 - \rho)\tau + \sum_{i=1}^{m} \Delta\tau_i$ (equation 4.5)
  $\mu^{t+1} :=$ for example, the messages of the ant with the highest goal function value
  $t := t + 1$
end
return B

Algorithm 6: The BP-ACO algorithm. The parameter $\rho \in [0, 1]$ is the evaporation rate of the pheromones and α and β control the influence of resp. the pheromones and the messages.

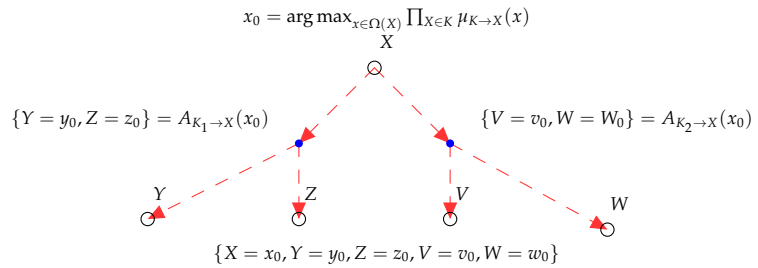


Figure 4.8: A display of the assignment retrieval phase. The set at the bottom is the complete assignment found by taking the union of all the partial assignments in the subtrees. K_1 is the clique containing $\{X, Y, Z\}$; K_2 is the clique containing $\{X, V, W\}$.

4.4 Comparison of BP-ACO and ACO for MPE Inference

ACO is an algorithm that only accepts Bayesian networks, while BP-ACO accepts the more general Markov networks as input. As every Bayesian network can be converted into a Markov network, BP-ACO accepts strictly more inputs, i.e. Markov networks which are not Bayesian networks. Bayesian Markov networks can be translated into Bayesian networks by demoralising the network.

ACO is much less complex than BP-ACO, and should be preferred for smaller Bayesian networks because of BP-ACO's higher initialisation costs. Of course, if the input is a non-Bayesian Markov network, only BP-ACO can be used.

4.5 Complexity

In this section we will try to give the complexities of the theoretical algorithms, not their implementations. It may however be necessary to fall back to the implementations, because some data structures are unspecified in the pseudocode and the operations of different data structures may have not have the same complexity. To study the complexity of the algorithms in this chapter it is useful to quantify the objects used in the algorithms using the following variables:

- a is the number of ants
- s is the number of iterations
- $n = \#\mathcal{E}$ is the number of edges
- $m = \#\mathcal{V}$ is the number of variables
- $m^+ = \#(\mathcal{V} \cup \mathcal{C})$ is the number of nodes in a Markov network
- $c = \#\mathcal{C}$ is the number of cliques

- $\ell = \max\{\#\Omega(X) : X \in \mathcal{V}\}$ is the maximum number of values any variable can assume
- $K = \max\{\#C : C \in \mathcal{C}\}$ is the largest clique size
- $L = \max\{\#\{X \in C : C \in \mathcal{C}\} : X \in \mathcal{V}\}$ the maximum number of cliques any variable can be contained in
- $P = \max\{\#\text{par}(X) : X \in \mathcal{V}\}$ is the highest number of parents a variable can have

Remember that the variables $\mathcal{V}$ and cliques $\mathcal{C}$ are both nodes in a Markov network, so the set of nodes of the network is given by $\mathcal{V} \cup \mathcal{C}$.

We assume that evaluating a Bayesian joint distribution costs $\mathcal{O}(Pm)$ time because there are m CPTs and looking up a value in a CPT costs $\mathcal{O}(P)$ steps. We further assume that evaluating a Markov joint distribution costs $\mathcal{O}(Kc)$ time because there are c cliques and if we want to look up a value in a factor we need to inspect $\mathcal{O}(K)$ values. Both algorithms have a factor sa in their complexity, because they both repeat s times and in each iteration a ants traverse the graph.

4.5.1 Ant colony optimisation for MPE Inference

A full graph traversal takes m steps and at every step the transition probabilities of ℓ outgoing edges need to be evaluated, in the worst case. At the end of an iteration we need to to evaluate the joint distribution for every ant, which costs $\mathcal{O}(Pm)$ time (see above). Then we deposit the created pheromones at $m-1$ edges for every ant. Note in particular that we can avoid iterating over all the edges by just iterating over the edges which will have a non-zero pheromone increase.

The complete complexity of ACO is therefore

$$\mathcal{O}(sa(m\ell + mP + m - 1)) = \mathcal{O}(sa((\ell + 2 + P)m - 1)) = \mathcal{O}(sa(\ell + P)m).$$

Hence, the ACO algorithm performs linearly in the number of network variables, m.

4.5.2 BP-ACO

At the beginning of each iteration, the transition probabilities for the spanning tree algorithm need to be calculated. There are n edges in the network and for every edge (i, j) we need to calculate 2 transition probabilities: p_{ij} and p_{ji}. Calculating the transition probabilities therefore costs $2n$ steps.

Then, for each ant, we calculate a spanning tree of the network using the transition probabilities. The outer loop in this algorithm (algorithm 3) consists of m^+ iterations and the inner backtracking loop consists of at most m^+ iterations. Hence, the SpanningTree algorithm takes $\mathcal{O}((m^+)^2)$ time in the worst case.

Subsequently we calculate the "missing edges": the edges which are in the network, but not in the spanning tree. This takes $\mathcal{O}(cK(m^+ - 1)) = \mathcal{O}(cKm^+)$ time: for each clique, we need to check for every of its edges if it is not contained in the spanning tree.

Then we arrive at the inward pull phase (algorithm 4), where the messages are updated from the leaves to the root. There are two types of messages which can be sent:

- Messages from a variable to a clique: These cost $\ell(L-1)$ steps to update in the worst case if we assume that every variable is in the same number of cliques. The ℓ factor originates from the domain of the message, which is at most as big as the maximum number of values of a variable in the network. The $L-1$ comes from the fact that a variable can be at most in L cliques, so for each message from a variable to a clique we need to multiply at most $L-1$ messages from different cliques to the same variable.

- Messages from a clique to a variable: These cost $\ell L^{K-1}K$ steps to update in the worst case if we assume that every clique is of the same size. The ℓ factor has the same meaning as in the other type of message. The L^{K-1} is derived from the fact that we need to maximise over the values of the other variables in the cliques. There are at most K variables in any clique, and any variable can assume at most L values, so we maximise over L^{K-1} elements. The last factor K is a result of taking the product of a Markov network factor (φ) and $K-1$ messages.

Assume, as a worst case scenario, that the i-th tree (with $1 \leq i \leq a$) has l_i leaves which are all variable nodes. Then we need to compute $m - l_i$ new variable-to-clique messages, which costs $(m-l_i)\ell(L-1)$ operations in total, and $m^+ - 1 - (m - l_i) = m^+ - m + l_i - 1$ new clique-to-variable messages for tree i, which costs $(m^+ - m + l_i - 1)\ell L^{K-1}K$ operations in total. Calculating all the messages therefore costs $\ell((m-l_i)(L-1) + (m^+ - m + l_i - 1)L^{K-1}K)$ operations in the worst case.

The next phase is the outward push phase (algorithm 5), where we collect partial variable assignments and join them together into one complete assignment for the network. There are two cases:

- The root node is a variable node. We need to take the maximum of at most ℓ values and then we need to traverse the remaining $m^+ - 1$ nodes of the tree. We assume that looking up an assignment at a clique node further down the tree is constant time. This means

- The root node is a clique node. We apply the algorithm recursively to the subtrees with the child variable nodes as root nodes. With similar reasoning as in the other case, there are now $m^+ - 1 + K\ell$ steps required in the worst case. The K factor is necessary because of the recursive call to the (at most) K subtrees.

In the worst case, this phase thus consists of $\max\{m^+ - 1 + \ell, m^+ - 1 + K\ell\} = m^+ - 1 + K\ell$ steps, because $K \geq 1$.

Finally, every ant needs to check if the assignment that was found in the previous phase is better than the best assignment so far. If we save the joint distribution of the best assignment separately, this requires c steps because of the evaluation of the retrieved assignment.

At the end of each iteration, the pheromone table needs to be updated. Doing this requires a complete assignments and spanning trees from the previous phase: one for each ant. For each ant, its assignment is evaluated using the joint distribution, which requires $\mathcal{O}(Kc)$ steps (see above). Then we deposit pheromones equal to the joint distribution value (equation 4.5) on the $m-1$ edges of the spanning tree. Updating the pheromones hence costs at most $a(Kc+m-1)$ steps in total each iteration.

Taking the complexities of all the separate sections of the algorithm together, results in a complete complexity of

$$\begin{aligned}
&\mathcal{O}\left(s\left(2n+a\left(\left(m^{+}\right)^{2}+cKm^{+}\right)+\sum_{i=1}^{a}\ell\left(\left(m-l_i\right)\left(L-1\right)+\left(m^{+}-m+l_i-1\right)L^{K-1}K\right)+a\left(Kc+m-1\right)\right)\right)\\
&=\mathcal{O}\left(s\left(n+a\left(\left(m^{+}\right)^{2}+cK(m^{+}+1)+c+m\right)+\ell\sum_{i=1}^{a}\left(\left(m-l_i\right)L+\left(m^{+}-m+l_i\right)L^{K-1}K\right)\right)\right).
\end{aligned}$$

If we now assume that the number of leaves of each spanning tree is a constant l, this gigantic expression reduces to

$$\begin{aligned}
&\mathcal{O}\left(s\left(n+a\left(\left(m^{+}\right)^{2}+cKm^{+}+c+m\right)+a\ell\left(\left(m-l\right)L+\left(m^{+}-m+l\right)L^{K-1}K\right)\right)\right)\\
&=\mathcal{O}\left(s\left(n+a\left(\left(m^{+}\right)^{2}+cKm^{+}+c+m+\ell\left(\left(m-l\right)L+\left(m^{+}-m+l\right)L^{K-1}K\right)\right)\right)\right).
\end{aligned}$$

The dominating section of the algorithm is the belief propagation section. If we assume that the number of iterations and ants is constant, this leaves us with a complexity of $\mathcal{O}(\ell\left(m^{+}-m+l\right)L^{K-1}K)$, so it is exponential in the largest clique size but also depends on the amount of edges because of the $m^{+}-m+l$ factor.

Chapter 5

Experiments

In Section 4.5 we offered a view on the theoretical performance of the algorithms. In this chapter we show how the algorithms perform in simulations.

The algorithms have been implemented with the non-strict[1] functional programming language *Haskell*, using GHC version 7.6.3. The ILP solver that was used for these experiments was SCIP, version 3.1.0 for $x86\_64$ Linux. The experiments were run on laptop with Debian Linux, version Jessie.

To find the parameters for the experiments we generally tweaked them a bit to be suitable for the benchmark at hand. The parameters α and β are equal to 1 and in all the experiments, but for clarity we have stated the values of these parameters in the caption of each of the experiments individually. The value 1 was chosen because these parameters had no observable influence on the benchmarks. In almost all experiments we chose $\rho = \frac{1}{10}$. Experiment 5.2 is the only experiment where we did not use this value, because changing this parameter has no influence on the execution time.

In this chapter, "the ILP algorithm" is understood to be the conversion of a Bayesian/Markov network to an integer linear program together with the execution of the ILP solver.

Not all of the datasets were used in every experiment, as some of them are too small and converge to an optimal solution too quickly, while others are too big and would cause the experiments to take too much time. Furthermore, for the rest of this chapter we will assume that BP-ACO uses the mean to calculate the heuristic value of a message.

[1] *Non-strictness* is a reduction strategy where the reduction of an expression is done from the highest level to the lowest level. This is in contrast to *strict* reduction, where every subexpression is reduced before its parent expression.
For example, let $f(\alpha, \beta) = \alpha$ for all α and β. Then $f(2, 3+5) \implies 2$ when reduced non-strictly, but $f(2, 3+5) \implies f(2, 8) \implies 2$ when reduced strictly.
If we denote the undefined expression by $\perp$ (it is of type $\forall \alpha \colon \alpha$, i.e. it is an element of every type), then we furthermore have $f(2+1, \frac{1}{0}) \implies 2+1 \implies 3$ when reduced non-strictly, but $f(2+1, \frac{1}{0}) \implies f(3, \perp) \implies \perp$ when reduced strictly.

Dataset	#Variables	#Edges (Bayes)	#Edges (Markov)	#Cliques	Max clique size	File size
fire_alarm	6	5	11	6	3	196B
spect	23	22	45	23	2	1606B
alarm2	37	46	83	37	5	4282B
emdec6g	168	261	429	168	6	17830B
cpcs54	54	108	162	54	10	19704B
diagnose_a	203	298	501	203	9	75235B
barley	48	84	132	48	5	1931828B

Figure 5.1: Metadata of the datasets which have been used in the experiments. The number of edges are given for both the Bayesian network structure as well as the Markov network structure. This is useful information, because the ant colony algorithm requires Bayesian networks while the other algorithms require Markov networks.

5.1 Execution time

This section explores the execution times for all algorithms. First, we compare the execution times of the approximation algorithms using the same parameters. In the second plot we investigate how much a lower bound reduces the execution time of the ILP solver.

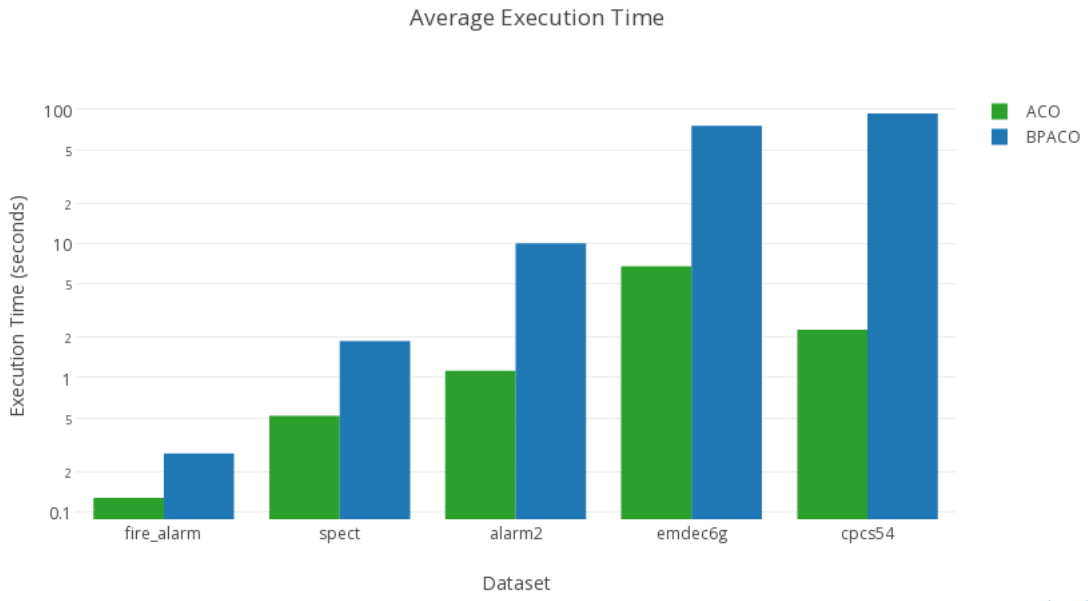


Figure 5.2: This bar plot displays the average execution time of the approximation algorithms on several datasets with common parameters $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{2}$, 10 iterations and 15 ants. The datasets are sorted by file size.
Note that ACO takes longer on *emdec6g* than *cpcs54*, because it has thrice as many variables and more than twice as many edges. (table 5.1)

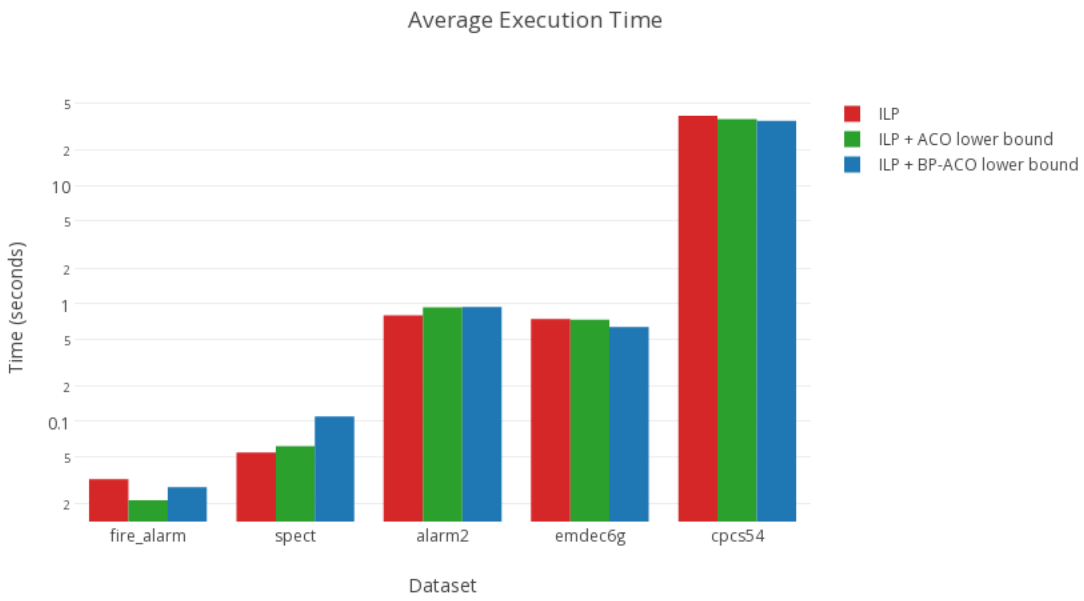


Figure 5.3: This bar plot displays the average execution time of the exact ILP solver on several datasets in three separate ways, using parameters $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$, 2 iterations and 1 ant. The red bars show the execution time of just the ILP algorithm, the green bars show the execution time of the ILP algorithm using the solution from ACO as a lower bound and the blue bars show the execution time of the ILP algorithm using the solution from BP-ACO as a lower bound.
The ACO and BP-ACO lower bounds do not lower the execution time of the ILP algorithm by much on the smaller datasets *fire_alarm* and *emdec6g* and even increase the execution time on *spect* and *alarm2*. The biggest dataset (*cpcs54*) however has an execution time decrease of 2.4 seconds using the ACO lower bound and a decrease of 3.5 seconds using the BP-ACO lower bound. The ILP algorithm without lower bound ran for 39 seconds on average.
We conclude from these results that it is worthwhile to run an approximation algorithm for a lower bound that can be used by the ILP algorithm if the dataset is big.
The datasets are sorted by file size (table 5.1).

5.2 Solution quality

In this section we analyse the degradation of the solution quality, i.e. how the solutions of the approximation algorithms degrade with respect to the real ILP solution as the dataset grows in size. We find that the solution quality indeed decreases fairly quickly as the dataset becomes bigger.

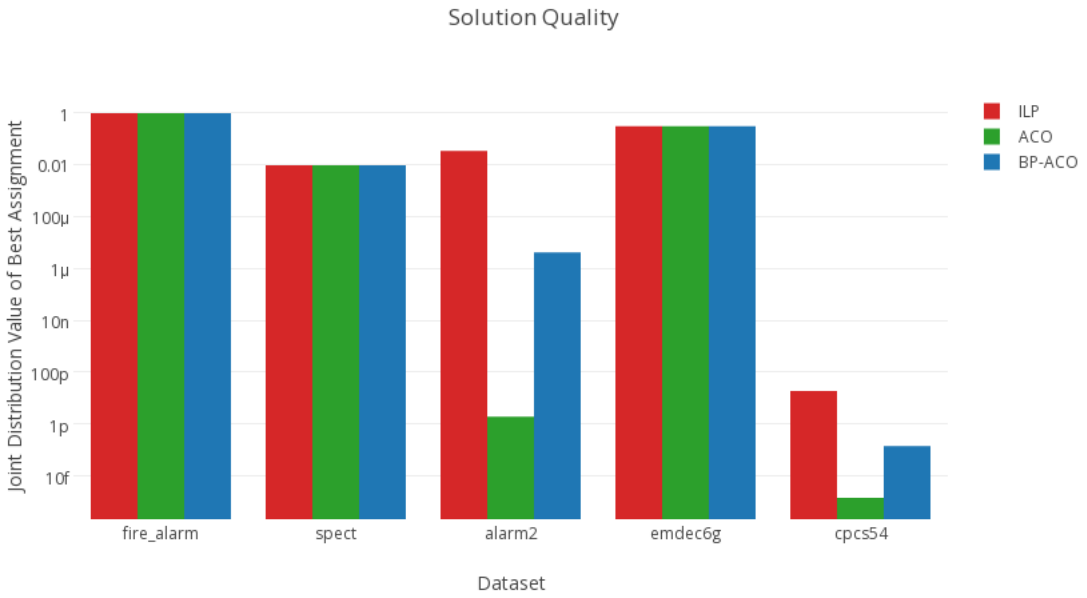


Figure 5.4: This bar plot shows the solution quality of the approximation algorithms compared to the exact ILP algorithm on several of the datasets given above. The parameters are $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$, 5 iterations and 5 ants.
From this plot we can extract that the solution quality degrades rapidly as the dataset grows in size. We can also see that BP-ACO produces much better solutions than ACO, but they were both given the same amount of iterations and ants even though BP-ACO puts much more work into each iteration compared to ACO. Hence, the solution qualities of BP-ACO and ACO should not be compared; this plot only serves to demonstrate the degradation of both solution qualities with respect to the real solution (given by the ILP algorithm).
The datasets are sorted by file size (table 5.1).

5.3 Convergence

In this section we will show how the approximation algorithms behave asymptotically. We have tested the algorithm convergence on three datasets: *cpcs54*, *diagnose_a* and *barley*. To arrive at the plots in this section we ran both algorithms 15 times and chose the best representative of the average execution of the respective algorithm.

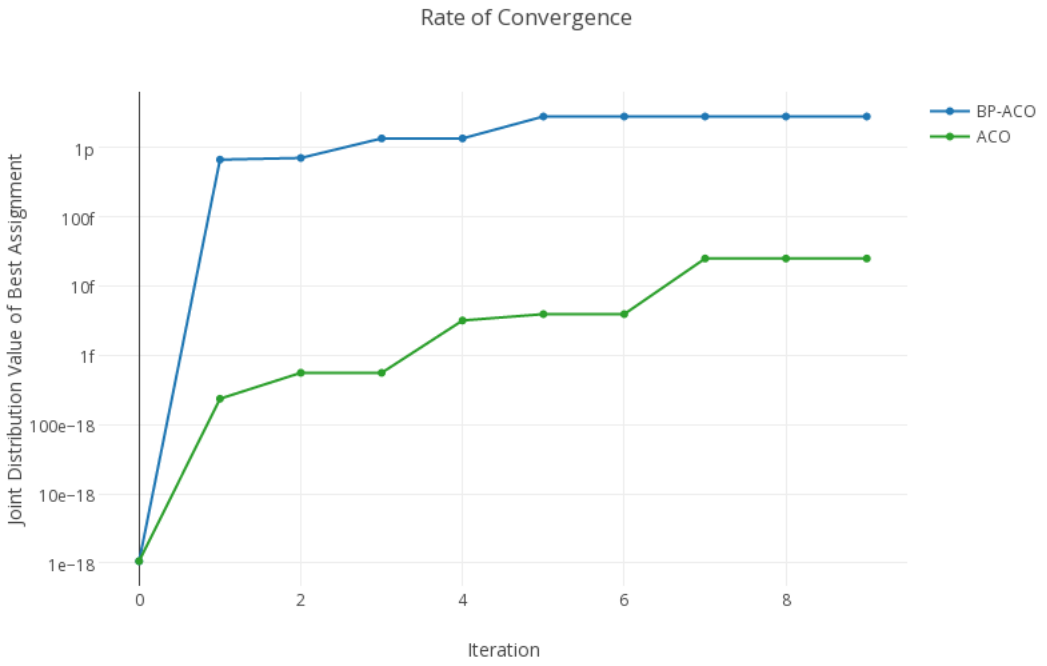


Figure 5.5: These graphs show the rate of convergence of the algorithms on the *cpcs54* dataset with respect to the iteration count. The parameters are $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$, 10 iterations and 3 ants.
Both algorithms immediately find a much better assignment in the first iteration, but in later iterations both algorithms improve slowly. BP-ACO produces notably better assignments than ACO, but performs much more work in each iteration and is therefore slower.

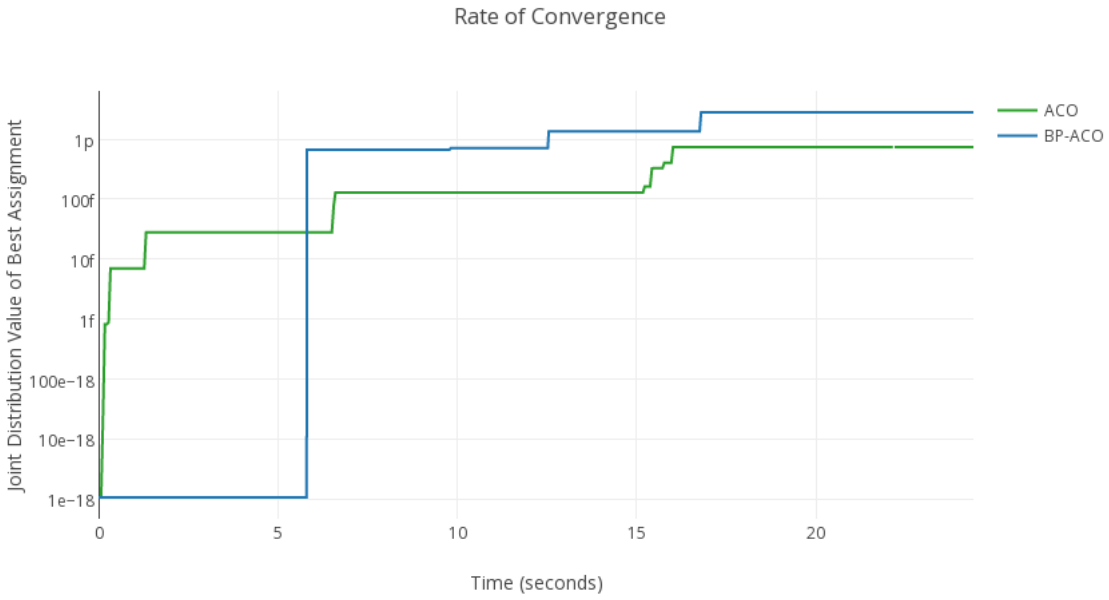


Figure 5.6: These graphs show the rate of convergence of the algorithms on the *cpcs54* dataset with respect to time. The parameters are $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$ and 3 ants.
In this benchmark, both algorithms ran for approximately 25 seconds, BP-ACO used 10 iterations whereas ACO used 464 iterations. In the first 5 seconds, BP-ACO performed badly in comparison with ACO. The reason for this is that BP-ACO's first iteration covered these first 5 seconds, and that there were still some initialisations to be done. Later iterations also benefit from sharing of data structures in Haskell, meaning that fewer unnecessary copies are done.
We can see that BP-ACO outperforms ACO not only if they are given the same number of iterations to execute (plot 5.5), but also the same amount of time.
The short period of time of 25 seconds was chosen because it corresponded with 10 iterations of BP-ACO, but also because after 25 seconds the graphs remained constant.

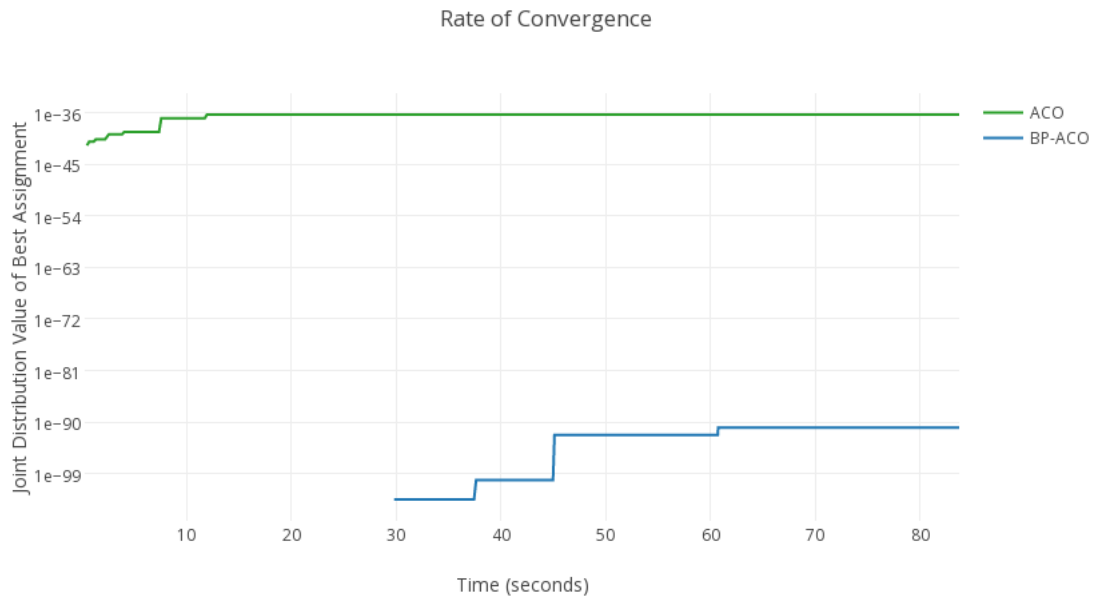


Figure 5.7: This line graph shows the rate of convergence of the algorithms on the *diagnose_a* dataset with respect to time. The parameters are $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$ and 3 ants.
In this benchmark, both algorithms ran for approximately 25 seconds. In comparison with plot 5.6, BP-ACO performs consistently worse than ACO.

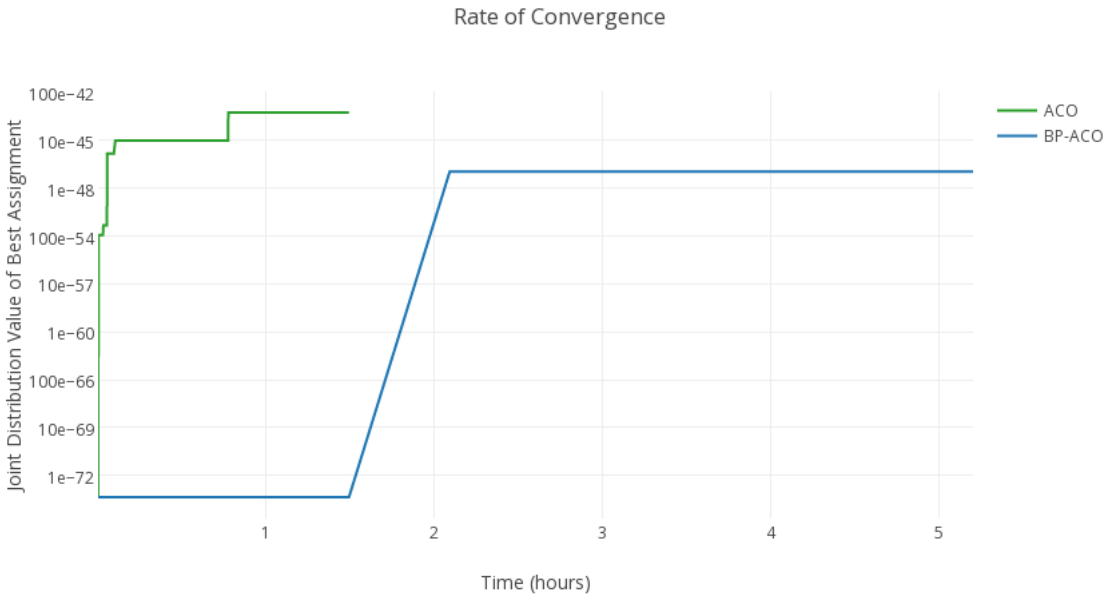


Figure 5.8: This line graph shows the rate of convergence of the algorithms on the *barley* dataset with respect to time. The common parameters are $\alpha = 1$, $\beta = 1$, $\rho = \frac{1}{10}$ and 3 ants. In addition, BP-ACO uses the mean to calculate the heuristic value of a message.
In this benchmark, BP-ACO ran for about 5 hours and 12 minutes while ACO only ran for about an hour and 30 minutes. ACO was not run for as long as 5 hours and 12 minutes because after running for $1\frac{1}{2}$ hours, the benchmark began to run slower and slower, possibly caused by a space leak in the implementation. However, the results are still clear: ACO managed to produce a solution which is several orders of magnitude better BP-ACO's solution, even 30 minutes before BP-ACO had found its first solution.

Chapter 6

Conclusion

The goal of this thesis was to research methods for performing MPE inference in Markov and Bayesian networks. To this end, we have defined three algorithms in this thesis, and given their complexities:

- A procedure for transforming the problem of MPE inference in Markov networks to an ILP problem. It is exponential in the maximum clique size, but is typically very fast.
- The BP-ACO algorithm, an MPE approximation algorithm which is a combination of belief propagation and ant colony optimisation. It is exponential in the maximum clique size.
- The ACO algorithm, which is also an MPE approximation and walks down an *assignment tree* to find its solutions. It is linear in the number of variables.

The ILP formulation of MPE inference allows for easy addition of extra constraints to the maximisation problem. We can add constraints based on the probability distribution of other networks over the same variables and add evidence (i.e. maximise given the knowledge that $A = a$).

The experiments in the previous section do not show a clear winner among the approximation algorithms. Currently ACO seems to fare better in general, but with more work on BP-ACO, it can potentially become nearly as fast as ACO.

The experiments do show that our idea of adding approximate solutions as lower bound constraint to the ILP versions of MPE inference does make the ILP solver faster.

6.1 Future Work

In this section we document some research directions which may be interesting to follow.

- Long-term behaviour analysis of ACO and BP-ACO. To get a theoretical understanding of the convergence of the algorithm, the algorithms must be analysed mathematically. Because of its complexity this will be a difficult task for BP-ACO, but it could be done for ACO. To analyse the long-term behaviour of ACO, we need to know how the pheromones evolve. For reference, the pheromones are updated as follows:

$$\tau^0 = \text{uniformly initialised}$$
$$\tau^{k+1} = (1-\rho)\tau^k + \Delta\tau^k = (1-\rho)^{k+1}\tau^0 + \sum_{i=0}^{k}(1-\rho)^{k-i}\Delta\tau^i(\tau^i)$$

  where the superscript k stands for the iteration counter in τ and $\Delta\tau$. We have that $\Delta\tau^i(\tau^i) = \Delta\tau^i$ because the value of $\Delta\tau^i$ is determined by the solutions which used the pheromones τ^i to traverse the tree. Note also that $\Delta\tau^i(\tau^i)$ is a stochastic term in the equation, hence the ant colony update equation gives rise to a stochastic recurrence relation.

- Something we have not considered in the implementations of the algorithms are networks which consist of independent components (subgraphs). In principal, the algorithms can be run on the separate components and the results can be merged together. Performing MPE inference in network consisting of n independent components is thus equivalent to performing MPE inference on each separate component.

- On the level of the ants, both BP-ACO and ACO can be parallelised because of the independence of the ants. On the level of belief propagation in BP-ACO, BP-ACO can be parallelised further because of the independence of the leaves. If the algorithms are optimised in this way, then it would be interesting to see new results from experiments, especially because there two places where BP-ACO can be parallelised whereas there is only one place for ACO.

- There are various ways in which we can further parametrise BP-ACO. We could for example let the root node also be chosen by the ant colony algorithm, and look into ways of combining the messages from the ants in each iteration.

  Currently, the messages that have been updated by the ants are discarded every iteration, but we can envisage scenarios where it would be a good decision to set the messages of the next iterations to the messages of an ant with the highest solution value.

Bibliography

[Dar09] Adnan Darwiche. Modeling and Reasoning with Bayesian Networks. 2009.

[GBH04] Haipeng Guo, Prashanth R. Boddhireddy, and William H. Hsu. An ACO algorithm for the Most Probable Explanation Problem. 2004.

[HC71] J.M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. 1971.

[Nie] Mathias Niepert. A Delayed Column Generation Strategy for Exact k-Bounded MAP Inference in Markov Logic Networks.

[Pea85] Judea Pearl. Bayesian Networks: A model of self-activated memory. 1985.

[YFW01] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding Belief Propagation and its Generalizations. 2001.