Optimal Triangulation of Bayesian Networks for Efficient Inference

Chao Li

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy in Engineering

GRADUATE SCHOOL OF INFORMATION SYSTEMS

THE UNIVERSITY OF ELECTRO-COMMUNICATIONS

MARCH 2017

Optimal Triangulation of Bayesian Networks for Efficient Inference

APPROVED BY SUPERVISORY COMMITTEE:

CHAIRPERSON:	Professor Maomi Ueno
MEMBER:	Professor Akihiko Ohsuga
MEMBER:	Professor Satoshi Kurihara
MEMBER:	Associate Professor Shuichi Kawano
MEMBER:	Professor Hiro Ito
MEMBER:	Associate Professor Yoshio Okamoto

ベイジアンネットワークにおける確率推論の高速化のための最適三角化アルゴリズムの提案

李　超

概要

ベイジアンネットワークは同時確率分布を厳密分解できる手法であり、多くの統計学習モデルの中でも最も予測精度が良いことが知られている。しかし、ベイジアンネットワークの確率推論はNP困難問題であり、既存アルゴリズムでは100変数程度しか扱うことができなかった。そこで、本研究では、ベイジアンネットワークにおける確率推論の高速化を目標とする。一般に、ベイジアンネットワークの確率推論は以下の手順で行う。1.ベイジアンネットワークを三角グラフ（コーダルグラフ）に変換,2.三角グラフから構成されるクリーク木に確率伝搬アルゴリズムを適用. ここで、三角グラフとは、長さが４以上のすべての閉路（サイクル）に少なくとも一個弦（コード）があるグラフである。ベイジアンネットワークのグラフ構造を三角グラフに変換することを三角化と呼ぶ。三角グラフでは、極大クリークの集合からクリーク木と呼ばれる特別な木構造を構築することができる。三角化により得られる三角グラフは手法によって異なるため、クリーク木における確率推論の計算量も三角化手法に依存する。したがって、確率推論の高速化のために、ベイジアンネットワークのグラフ構造を確率推論の計算量が小さい三角グラフを探索する必要がある。しかし、従来研究されてきた三角化手法をベイジアンネットワークに直接用いても最適値は保証されない。そこで、本研究では、推論アルゴリズムの計算量を示すTotal table size（ＴＴＳ）を三角化の最適基準とし、高速な最適三角化アルゴリズムを提案する。

具体的な三角グラフの構築にはノード消去法を用いる。まず、ベイジアンネットワークをモラルグラフに変換し、このモラルグラフから、逐次ノードを消去する。ここで、モラルグラフとは、共通の子ノードを持つノード間にエッジを加え、有向エッジの向きを取り除いたグラフである。モラルグラフから一つのノードを消去する時に、隣接ノード間にエッジを入れる。全てのノードを消去することにより、入れたエッジ集合とモラルグラフのエッジ集合の和集合により、三角グラフが構築できる。このとき、ノード消去の順序により得られた三角グラフとＴＴＳが異なるものになる。これまでに三角化を最適化するために深さ優先探索を用いることが提案されている。しかし、深さ優先探索におけるノード消去順のパターン数はN!（Nは変数の数）であるため、変数数が多い（１００程度）ベイジアンネットワークに対しては最適三角化の計算が困難になる。そこで、本研究では、既存の深さ優先手法を改善したアルゴリズムを提案する。具体的には、次の2つのアルゴリズムを提案する。1）Dynamic clique maintenanceアルゴリズム：エッジ追加によって生成される新たなクリークの列挙を高速化するために，新しく追加されるエッジの共通隣接ノードのみからクリークを列挙するアルゴリズム，2）Pivot clique pruningアルゴリズム：ＴＴＳを最小化する三角グラフの深さ優先探索の探索空間中に，任意の分枝について必ず同一の三角グラフを示す異なる分枝

が一つ以上存在する性質を利用することで、探索空間を大幅に軽減するアルゴリズム.

さらに、本研究では、様々な実験により、従来の手法に比べて、提案手法が高速に最小のＴＴＳを持つ三角グラフを探索できることを示すとともに、最小ＴＴＳを最適基準とする提案手法が、従来の最適基準を適用する手法より有意に小さい計算量の三角グラフを発見できることを示した。

ABSTRACT

Optimal triangulation of Bayesian networks
for efficient inference
by
Chao Li
Doctor of Philosophy in Engineering
The University of Electro-Communications
Chairperson: Professor Maomi Ueno

Bayesian networks are widely used probabilistic graphical models that provide a compact representation of joint probability distributions over a set of variables. A common inference task in Bayesian networks is to compute the posterior marginal distributions for the unobserved variables given some evidence variables that we have already observed. However, the inference problem is known to be NP-hard and this complexity of inference limits the usage of Bayesian networks. Many attempts to improve the inference algorithm have been made in the past two decades. Currently, the junction tree algorithm is among the most prominent exact inference algorithms. To perform efficient inference on a Bayesian network using the junction tree algorithm, it is necessary to find a triangulation of the moral graph of the Bayesian network such that the total table size is small. In this context, the total table size is used to measure the computational complexity of the junction tree inference algorithm. This thesis focuses on exact algorithms for finding a triangulation that minimizes the total table size for a given Bayesian network.

For optimal triangulation, Ottosen and Vomlel have proposed a depth-first search (DFS) algorithm. They also introduced several techniques to improve the

DFS algorithm, including dynamic clique maintenance and coalescing map pruning. Nevertheless, the efficiency and scalability of their algorithm leave much room for improvement. First, the dynamic clique maintenance allows the recomputation of some cliques. Second, for a Bayesian network with n variables, the DFS algorithm runs in $\mathcal{O}^*(n!)$ time because it explores a search space of all elimination orders. To mitigate these problems, an extended depth-first search (EDFS) algorithm is proposed in this thesis. The new EDFS algorithm introduces two techniques: (1) a new dynamic clique maintenance algorithm that computes only those cliques that contain a new edge, and (2) a new pruning rule, called pivot clique pruning. The new dynamic clique maintenance algorithm explores a smaller search space and runs faster than the Ottosen and Vomlel approach. This improvement can decrease the overhead cost of the DFS algorithm, and the pivot clique pruning reduces the size of the search space by a factor of $\mathcal{O}(n^2)$. Our empirical results show that our proposed algorithm finds an optimal triangulation markedly faster than the state-of-the-art algorithm does.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Maomi Ueno for all of his support during my PhD studies and for encouragement in completing this thesis. I especially thank him for always leading me in the right direction and for bringing me into the Bayesian networks community. I am also very grateful for his constant help in my writing, for help in revising my papers and discussing my studies.

Besides my advisor, I would like to thank the rest of my thesis committee, Professor Ito, Professor Kawano, Professor Kurihara, Professor Ohsuga, and Professor Okamoto, not only for their insightful comments and encouragement, but also for their hard questions, which prompted me to widen the perspective of my research.

I also appreciate the important work of many researchers from the Bayesian networks community, including Joe Suzuki, Shin-ichi Minato and Jiri Vomlel, who encouraged my studies and provided valuable comments on my previous work. In particular, I am grateful to Dr. Thorsten Ottosen for his helpful comments and a discussion at PGM2012 about pivot clique pruning.

I would like to thank the many people who engaged and supported me during my graduate studies at The University of Electro-Communications. Last but not least, I would like to thank my parents for supporting me spiritually throughout writing this thesis and my life in general.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Bayesian networks are graphical models that encode probabilistic relations among several variables [Pearl, 1988]. A Bayesian network is a directed acyclic graph in which vertices represent random variables and the arcs (or lack of them) represent the direct dependence (or conditional independence) relations between the variables. Each network variable is associated with a conditional probability table conditioning on its parent variables, which quantifies the relation between the variable and its parents. Overall, a Bayesian network provides a compact representation of joint probability distributions over the network variables.

An extremely common inference task in Bayesian networks is to compute the posterior marginal distributions for the unobserved variables given some evidence variables that we have already observed. However, exact computation of posterior marginal distributions in a Bayesian network is known to be NP-hard [Cooper, 1990] and even an approximation of them is computationally intractable in the general case [Roth, 1996]. Consequently, the inference algorithm has a network size limitation that hinders the more widespread application of Bayesian networks. Many attempts to improve the inference algorithm have been made in the past two

decades. The junction tree algorithm [Lauritzen and Spiegelhalter, 1988; Jensen et al., 1990; Shenoy and Shafer, 1990] is currently among the most prominent exact inference algorithms. In that algorithm, a Bayesian network is first converted into a special data structure called a junction tree, and then belief is propagated on the tree. A junction tree can be formed if and only if the moral graph of the Bayesian network is a chordal graph, also known as a triangulated graph. If the graph is not chordal, then extra edges should be added to it until it becomes so. The process of adding edges to a graph in order to make it chordal is called "triangulation" in the Bayesian networks field. In general, a Bayesian network allows several different triangulations, and the triangulation will affect the structure of the junction tree and the performance of subsequent belief propagation on that tree. Hence, to enable efficient inference on a Bayesian network using the junction tree algorithm, this study aims to find a triangulation of the moral graph of the Bayesian network such that the total table size is minimized [Kjaerulff, 1990; Nielsen and Jensen, 2007]. In this context, the total table size is used to measure the computational complexity of the junction tree inference algorithm. Unfortunately, finding a triangulation with the minimum total table size is known to be NP-hard [Wen, 1990]. Due to this complexity, early research in this direction focused mainly on developing approximation algorithms, such as greedy heuristics [Kjaerulff, 1990; Wen, 1990]. Heuristic approaches are useful for triangulation of large-scale Bayesian networks, for which finding an optimal triangulation is infeasible; however, these approximation methods are not guaranteed to find an optimal triangulation. Finding an optimal triangulation requires additional computational time, but once the junction tree of a Bayesian network has been constructed, efficient probabilistic inference can be performed on the same junction tree to process any evidence [Madsen and Jensen, 1999; Darwiche, 2009]. Therefore, an optimal triangulation can be found off-line and saved for use in inference algorithms. An additional reason to find an optimal triangulation is that performing inference on

Bayesian network systems with real-time computing constraints (including in real-time systems [Musliner et al., 1995] and embedded systems [Ramos and Cozman, 2005]) requires an optimal triangulation to minimize the inference time. Therefore, this thesis focuses especially on algorithms for optimal triangulation of Bayesian networks.

In order to construct an efficient junction tree, previous triangulation algorithms have used depth-first search [Gogate and Dechter, 2004], branch and bound [Bachoore and Bodlaender, 2006], best-first search [Dow and Korf, 2007] and dynamic programming [Bodlaender et al., 2012]. Instead of using the total table size as a measure, these methods have employed the treewidth criterion. The treewidth of a chordal graph is the size of the maximum clique minus one, and the treewidth criterion requires finding a chordal graph that has minimum treewidth. A junction tree is constructed by connecting the (maximal) cliques of a chordal graph. The complexity of belief propagation for a clique is proportional to the table size of the clique, which is the size of the joint state space of the variables represented by the vertices in the clique. The total computational cost of belief propagation is proportional to the total table size of the junction tree. For example, when we have a Bayesian network in which all variables have at most c states, the running time of the belief propagation using a junction tree with m cliques and treewidth k is of order $\mathcal{O}(c^k \cdot m)$. However, in practice, the statistical variables in a Bayesian network might have different numbers of states, and so a triangulation with minimum treewidth might not be optimal for this algorithm. Thus, the weighted treewidth is employed for triangulation algorithms, where the weighted treewidth of a chordal graph is the maximum table size required for any clique. Given a junction tree with m cliques and weighted treewidth w, the running time of a belief propagation is of order $\mathcal{O}(w \cdot m)$. Taking advantage of considering the different number of states over variables, the weighted treewidth criterion can obtain a better bound for inference time than the treewidth criterion. Several triangulation algorithms

that minimize the weighted treewidth have been proposed previously [Bachoore and Bodlaender, 2007; van den Eijkhof et al., 2007]. Nevertheless, when cliques are not almost all equal in table size (or, equivalently, weighted clique size), the time bound for the inference algorithm is loose. Finally, the total table size is the sum of all weighted clique sizes, and the total table size is proportional to the running time of junction tree inference. Of all these optimality criteria, the total table size yields the most exact bound for the time requirement of probabilistic inference [Lauritzen and Spiegelhalter, 1988; Wen, 1990; Nielsen and Jensen, 2007]. Therefore, for efficient inference on a Bayesian network, a triangulation is optimal when it has the minimum total table size.

A triangulation can be found by an elimination algorithm known as the Elimination Game, proposed by Parter [1961]. In this algorithm, a chordal graph is obtained by eliminating all vertices from a graph according to a linear ordering of the vertices of the graph (called the elimination order). Ottosen and Vomlel [2012] have shown that the optimal triangulation problem can be formulated as a problem to find an elimination order such that the chordal graph obtained according to the order has the minimum total table size. Employing this formulation, Ottosen and Vomlel investigated depth-first search and best-first search algorithms for exploring the search space of all elimination orders [Ottosen and Vomlel, 2012]. They claimed that depth-first search uses less memory than best-first search. Moreover, they demonstrated that the two methods have almost equal run times in computational experiments: that is, the best-first search, which theoretically has better order, does not necessarily run faster than the depth-first search in practice because, although the depth-first search expands more search nodes than the best-first search does, the best-first search has the heavy overhead of maintaining a priority queue. (To avoid confusion, in this paper, "vertex" is used exclusively in the context of the graph being triangulated and "node" is used exclusively in reference to the search space of the optimal triangulation algorithm.) This thesis

focuses mainly on improvements to depth-first search algorithms for optimal triangulation. In the depth-first search algorithm, in order to employ branch and bound for pruning, it is necessary to compute the total table size of each node, which is a lower bound for the node. To compute this quantity, we need to know the set of cliques of the graph to which each node belongs. A simple method for computing this is to run the Bron–Kerbosch (BK) algorithm [Cazals and Karande, 2008] for each node of the graph; however, the complexity of the BK algorithm is exponential in the number of vertices of the graph [Tomita et al., 2006]. To resolve this problem, Ottosen and Vomlel proposed a dynamic clique maintenance algorithm [Ottosen and Vomlel, 2012] that runs the BK algorithm on a smaller subgraph in which all the new cliques can be found and all known cliques within the subgraph are removed. This dynamic clique maintenance reduced the overhead cost of each node and made the optimal triangulation algorithm faster. To reduce the search space, Ottosen and Vomlel also introduced the simplicial vertex rule [Bodlaender et al., 2005; van den Eijkhof et al., 2007] and coalescing map pruning [Dow and Korf, 2007; Darwiche, 2009]. Nevertheless, the depth-first search algorithm proposed by Ottosen and Vomlel has the following two performance problems. First, the dynamic clique maintenance algorithm allows recomputing some cliques. The computational cost of the method increases with the number of duplicate computations. In the elimination process for triangulating a graph, it is well known that a new added edge cannot connect to a vertex that has been eliminated. From this observation, Li and Ueno [Li and Ueno, 2012] proposed an improved dynamic clique maintenance algorithm. The Li and Ueno method reduced the search space of the BK algorithm by removing eliminated vertices from the subgraph explored during the Ottosen and Vomlel method. However, this method still computes many duplicate cliques. Second, the depth-first search algorithm explores a search space of size $n!$, where n is the number of variables in the Bayesian network, because it explores the search space containing all elimination orders. It is known that some different

elimination orders induce identical triangulations. Consequently, the depth-first search algorithm might explore a large number of equivalent elimination orders.

1.2 Contributions

An extended depth-first search algorithm for the optimal triangulation of Bayesian networks is proposed. This algorithm improves the Ottosen and Vomlel method in two ways.

1. It reduces the overhead cost of each node, and
2. it reduces the size of the search space by a factor of $\mathcal{O}(n^2)$.

To reduce the overhead cost, this study proposes a new dynamic clique maintenance algorithm. When new edges are inserted in a graph during triangulation process, we need to update the stored cliques to be those of the new graph. Any new clique in the updated graph contains at least one new edge, and employing this observation in our method allows not recomputing those cliques that do not contain a new edge. Next, the proposed method runs the BK algorithm on the subgraph that contains only the vertices connected by new edges and all neighboring vertices of new edges. The proposed method, therefore, explores an even smaller subgraph than the one that the Ottosen and Vomlel method explores. Since the computational cost of dynamic clique maintenance is inherent in expanding each node, improving dynamic clique maintenance can decrease the overhead of each node. To reduce the size of the search space, a novel pruning rule, called pivot clique pruning, is introduced. The initial search space of the optimal triangulation algorithm includes all elimination orders; pivot clique pruning removes a large number of equivalent elimination orders from this search space. In a theoretical analysis, this paper shows that the pruning method reduces the size of the search space by a factor of $\mathcal{O}(n^2)$. Our empirical results show that the proposed depth-

first search algorithm represents a remarkable improvement over the Ottosen and Vomlel algorithm.

1.3 Overview of the rest of the thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the junction tree inference algorithm and motivates optimal triangulation algorithms. We also show that a recently proposed optimal triangulation algorithm has duplicate computation, whereas the algorithms in chapters 3 and 4 can mitigate this problem.

Chapter 3 proposes a new dynamic clique maintenance algorithm and evaluates the proposed method against the state-of-the-art method. The empirical results show that the new algorithm reduces the number of duplicate cliques computed during the triangulation process and remarkably improves the running time of the depth-first search algorithm.

Chapter 4 proposes pivot clique pruning. We first introduce the concept of equivalent elimination orders and then employ this observation to formalize the idea of pivot clique pruning. We also discuss the pivot clique choice heuristic and show how pivot clique pruning reduces the size of the search space by a factor of $\mathcal{O}(n^2)$.

Chapter 5 gives final conclusions and presents plans for additional research in the future.

Chapter 2

Triangulation of Bayesian networks

2.1 Background

A Bayesian network is a directed acyclic graph (DAG) in which the set of vertices corresponds to a set of (discrete) random variables $X = \{x_1, x_2, \ldots, x_n\}$, and the arcs represent direct dependency relations between the variables. For example, Figure 2.1 shows the classical Asia Bayesian network [Jensen et al., 1990]. More precisely, each variable x_i in X is represented as a vertex in the DAG and is associated with a conditional probability table (CPT), $P(x_i \mid PA_i)$, where PA_i denotes the parents of x_i in the DAG. Given a variable x_i in a DAG G, the parents of x_i in G are defined to be the set of variables with an arc to x_i. The product of CPTs in a Bayesian network gives the joint probability distribution of variables in the Bayesian network, with

$$P(X) = P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid PA_i), \tag{2.1}$$

where n is the number of variables in the Bayesian network.

When an inference task is performed on a Bayesian network, we typically com-

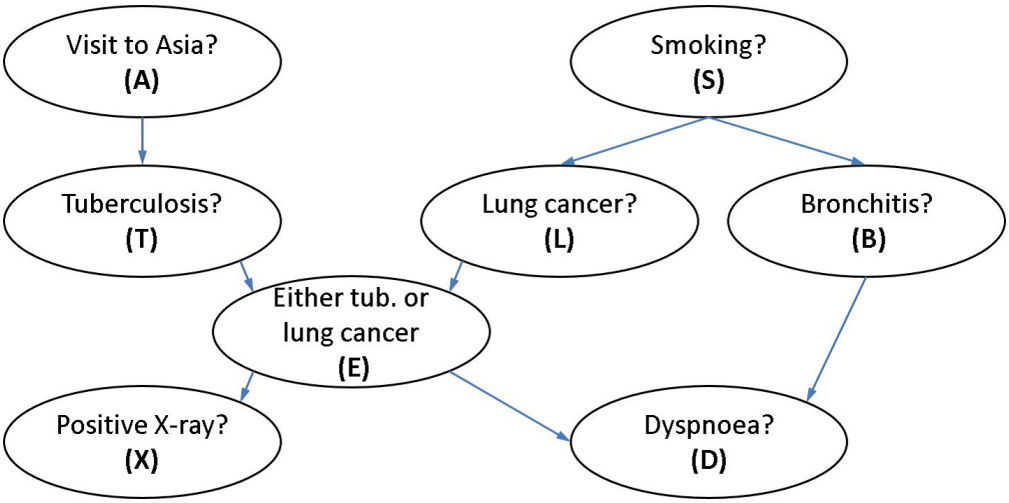


Figure 2.1: The Asia Bayesian network

pute the posterior marginal distributions for the unobserved variables given some evidence variables that we have already observed. However, computing the posterior marginal distributions is known to be NP-hard. Currently, the most efficient algorithm used for computing this distribution is the junction tree algorithm. The junction tree algorithm uses two processes: compilation and propagation. The compilation part of the method consists of the following steps:

1. moralize the Bayesian network graph, see Figure 2.2;
2. triangulate the moralized graph (i.e., add extra edges such that every cycle of length greater than three has a chord), see Figure 2.3a;
3. identify all maximal cliques of the chordal graph (a clique is defined as a subset of vertices of an undirected graph such that every two distinct vertices in the vertex subset are adjacent);
4. construct a junction tree over these cliques, see Fig. 2.3b.

A junction tree over the cliques is characterized by the junction tree property: given two cliques in the junction tree, C_i and C_j, every node on the path between

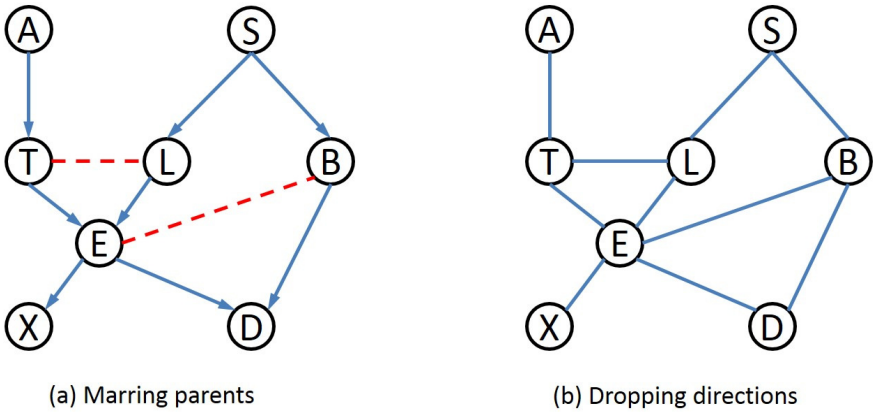


Figure 2.2: Moralizing the Bayesian network graph: (a) connect the vertices with common children and (b) drop the directions of directed edges.

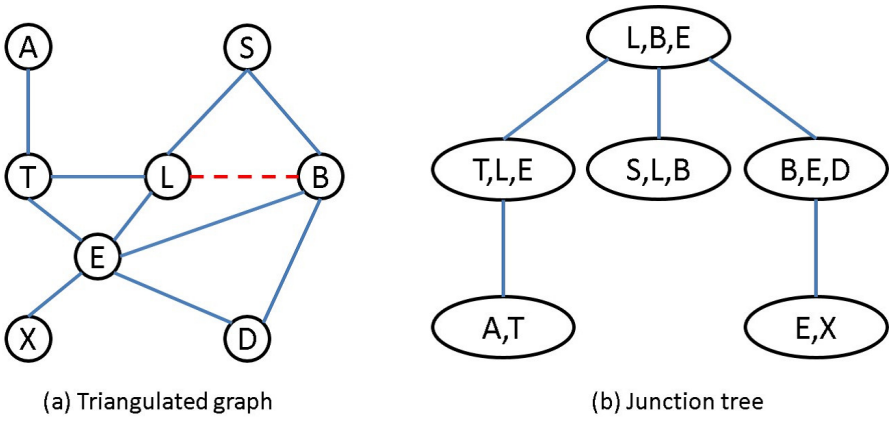


Figure 2.3: (a) Add edges to make the moral graph chordal and (b) construct a junction tree by connecting the cliques of the chordal graph.

them contains their intersection ($C_i \cap C_j$). In the compilation part, steps 1 and 3 are deterministic but steps 2 and 4 raise optimization problems. For step 2, we will discuss the optimal triangulation problem in detail in the next section. This thesis focuses on the optimal triangulation algorithms. For step 4, Jensen [Jensen and Jensen, 1994] has proposed an algorithm for optimal junction tree construction.

Before we discuss belief propagation, we must first introduce its central component: the potential. A potential ϕ is a function over a set of variables, mapping each instantiation of these variables to a non-negative number. Two potentials can be multiplied and divided. The marginalization operation is also well defined for a potential. A good introduction to these concepts can be found in [Nielsen and Jensen, 2007].

The propagation part of the method consists of the following steps:

1. giving all links in the junction tree a label consisting of the intersection of the neighboring cliques (these labels are called separators, see Figure 2.4a);

2. forming a potential ϕ_i for each clique C_i, using the CPTs of the Bayesian network and attaching a potential ϕ_{ij} for each separator with all values initialized to one; and

3. letting the nodes communicate via the separators. For example, in Figure 2.4b, sending a message from clique C_i to C_j with separator S_{ij} does the following. It computes a new potential $\phi'_{ij} \leftarrow \sum_{C_i \setminus S_{ij}} \phi_i$. It computes a message for node j : $M_{ij} = \phi'_{ij}/\phi_{ij}$ and multiplies the potential at node j by the computed message: $\phi_j \leftarrow \phi_j M_{ij}$. Finally, it replaces the potential ϕ_{ij} on the separator with ϕ'_{ij}.

Belief propagation begins by choosing an arbitrary clique as the root, from which the propagation is initiated. Message passing starts from the leaves and is divided into two stages. When a clique receives messages from all its neighbors except one that lies toward the root, it is allowed to send a message toward the root.

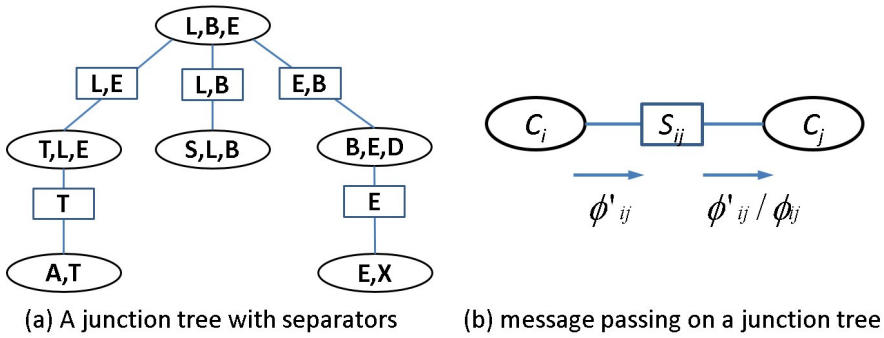


Figure 2.4: (a) Junction tree and (b) an illustration of message passing

This continues until the root clique has received messages from all its neighbors. This procedure is called COLLECT-EVIDENCE. Then, the root clique sends messages to all its neighbors. When a clique receives messages from all its neighbors, it sends a message toward the leaves until all leaves have received a message. This procedure is called DISTRIBUTE-EVIDENCE. After these two rounds of message passing, each clique potential of the junction tree holds the marginal probability distribution for the variables belonging to it.

Given a junction tree with m cliques and assuming only binary variables, performing probabilistic inference on the tree needs to calculate $\sum_{i=1}^{m} 2^{m_i}$ parameters, where m_i denotes the number of variables in the ith clique. The number $\sum_{i=1}^{m} 2^{m_i}$ is known as the total table size (or total clique tree size [Mengshoel, 2010] or total state space size [Kjaerulff, 1990]), and is an estimation of the time complexity of the junction tree algorithm. We will give a formal definition of total table size in the next section. A Bayesian network allows several different triangulations, yielding different sets of cliques. The time complexity of belief propagation heavily depends on the total table size of the chordal graph. Therefore, it is necessary to find an optimal total table size triangulation for efficient inference.

2.2 The triangulation problem

We first introduce some notation and definitions for the description of the triangulation problem. Then we formulate the search space of the optimal triangulation algorithm.

2.2.1 Notation and definitions

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E, $\mathcal{V}(G)$ denotes the vertex set of G and $\mathcal{E}(G)$ denotes the edge set of G. For a set of vertices $W \subseteq V$, $G[W] = (W, \{(v, w) \in E \mid v, w \in W\})$ is the subgraph of G induced by W. For a set of edges F, $\mathcal{V}(F)$ denotes the set of vertices $\{v, w|(v, w) \in F\}$. Two vertices v and w in $G = (V, E)$ are said to be adjacent if $(v, w) \in E$. The set of neighbors of v in graph $G = (V, E)$ is denoted by $\mathcal{N}(v, G) = \{w \in V|(v, w) \in E\})$. The family $\mathcal{FA}(U, G)$ of a set of vertices $U \subseteq V$ is defined as the set $(\bigcup_{u \in U} \mathcal{N}(u, G)) \cup U$.

A graph G is complete if all pairs of vertices $(u, v)(u \neq v)$ are adjacent in G. A set $W \subseteq V$ of vertices is complete in G if $G[W]$ is a complete graph. If W is a complete set and no complete set U exists such that W is a proper subset of U, then W is a *clique*. (Remark: Any complete set is called a clique in some of the literature. In that case, what we have defined as a clique is called a maximal clique.) The set of all cliques of graph G is denoted by $\mathcal{C}(G)$. Let $G' = (V, E \cup F)(F \cap E = \emptyset)$ be the graph obtained by adding a set F of new edges to $G = (V, E)$. Then, $\mathcal{RC}(G, G') = \mathcal{C}(G) \backslash \mathcal{C}(G')$ denotes the set of removed cliques, and $\mathcal{NC}(G, G') = \mathcal{C}(G') \backslash \mathcal{C}(G)$ denotes the set of new cliques. For example, in Fig. 2.5, let G be the graph on the left, and G' be the graph obtained by adding a new edge (c, d) to G. In this example, we can compute $\mathcal{C}(G) = \{\{a, b, c\}, \{b, d\}, \{d, e\}, \{c, e\}\}$ and $\mathcal{C}(G') = \{\{a, b, c\}, \{b, c, d\}, \{c, d, e\}\}$. Then we have $\mathcal{RC}(G, G') = \{\{b, d\}, \{d, e\}, \{c, e\}\}$ and $\mathcal{NC}(G, G') = \{\{b, c, d\}, \{c, d, e\}\}$.

Let $G = (V, E)$ be an undirected graph with vertex set V that corresponds to the variable set of a Bayesian network. The table size of a clique C in G is defined as $ts(C) = \prod_{(v \in C)} |sp(v)|$, where $sp(v)$ denotes the state space of the network variable corresponding to v. The total table size (tts) of a graph G is defined as $tts(G) = \sum_{C \in \mathcal{C}(G)} ts(C)$.

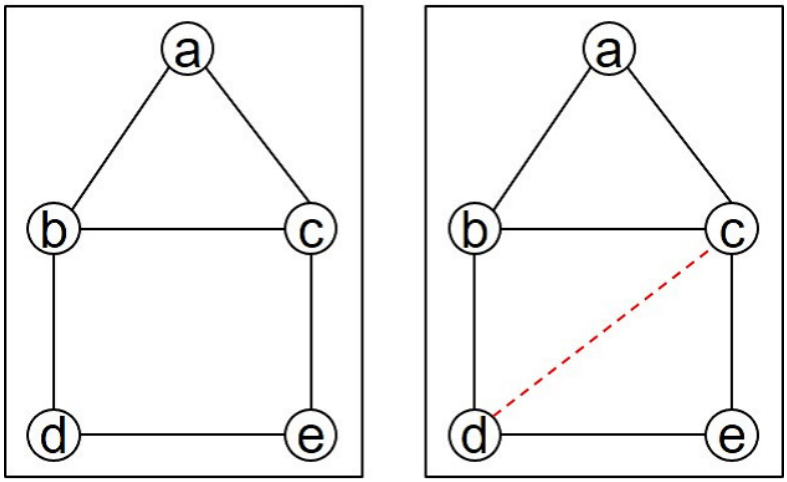


Figure 2.5: Left: Initial graph $G = (V, E)$. Right: Updated graph G' obtained by adding one edge (c, d) to G.

An undirected graph G is chordal if every cycle of length greater than three has a chord, that is, an edge connecting two nonconsecutive vertices in the cycle. Triangulation of $G = (V, E)$ is defined as adding a set of edges T such that $T \cap E = \emptyset$ and graph $H = (V, E \cup T)$ is chordal. For example, in Figure 2.5, the graph on the left is not chordal because a chord-less cycle $\{b, c, e, d\}$ exists. The graph on the right is chordal, and the edge (c, d) produces a triangulation for the graph on the left.

The elimination of a vertex $v \in V$ from graph $G = (V, E)$ is the process of adding necessary edges F to make the vertex set $\mathcal{N}(v, G)$ complete, and then removing v and its incident edges from G. The edges F added during the elimination

process are called fill-in edges. If $F = \emptyset$, then v is called a simplicial vertex of G. An elimination order for graph $G = (V, E)$ is a bijection $\pi\colon \{1, 2, \ldots, |V|\} \to V$ describing an order for eliminating all vertices from G, where $\pi(i)$ denotes the ith vertex in the order π. The elimination of vertices from graph G according to order π induces a remaining graph sequence $G_1^\pi, G_2^\pi, \ldots, G_n^\pi$, where graph $G_1^\pi = G$ and graph G_{i+1}^π is obtained by eliminating vertex $\pi(i)$ from graph G_i^π. Moreover, the elimination process induces a sequence of fill-in edges $F_1^\pi, F_2^\pi, \ldots, F_n^\pi$, where F_i^π are the fill-in edges introduced when eliminating vertex $\pi(i)$ from G_i^π. Let T^π denote the union of all the fill-in edges that result from eliminating all vertices from graph $G = (V, E)$ according to order π and let $H^\pi = (V, E \cup T^\pi)$ denote the filled-in graph that results from adding edges T^π to G. It is well known that T^π is a triangulation of G, and H^π is a chordal (or triangulated) graph [Golumbic, 2004]. The partially triangulated graph H_i^π for a graph G is defined as the graph that results from adding fill-in edges $F_1^\pi, F_2^\pi, \ldots, F_i^\pi$ to graph G. The final partially triangulated graph H_n^π (also written as H^π) is a chordal graph. Let τ denote a partial elimination order for graph G, which is a sequence of vertices for ordering the elimination process. The partially triangulated graph H^τ and the remaining graph G^τ are defined similarly.

Now, we present an example to demonstrate the process of eliminating vertices from the moral graph of the Asia Bayesian network in Figure 2.6. Consider an elimination order π starting with the sequence $\langle D, S\rangle$. Because eliminating vertex $\pi(1) = D$ does not add any fill-in edges, F_1^π is empty and D is a simplicial vertex. This process induces two associated graphs: a partially triangulated graph H_1^π (see Figure 2.6(a)) and the remaining graph G_2^π (see Figure 2.6(b)). Then we eliminate vertex $\pi(2) = S$. Eliminating vertex S adds a fill-in edge (L, B), so $F_2^\pi = \{(L, B)\}$. This process also induces two associated graphs: the partially triangulated graph H_2^π is shown in Figure 2.6(c) and the remaining graph G_3^π is shown in Figure 2.6(d). If we continue to eliminate vertices until no vertex is left,

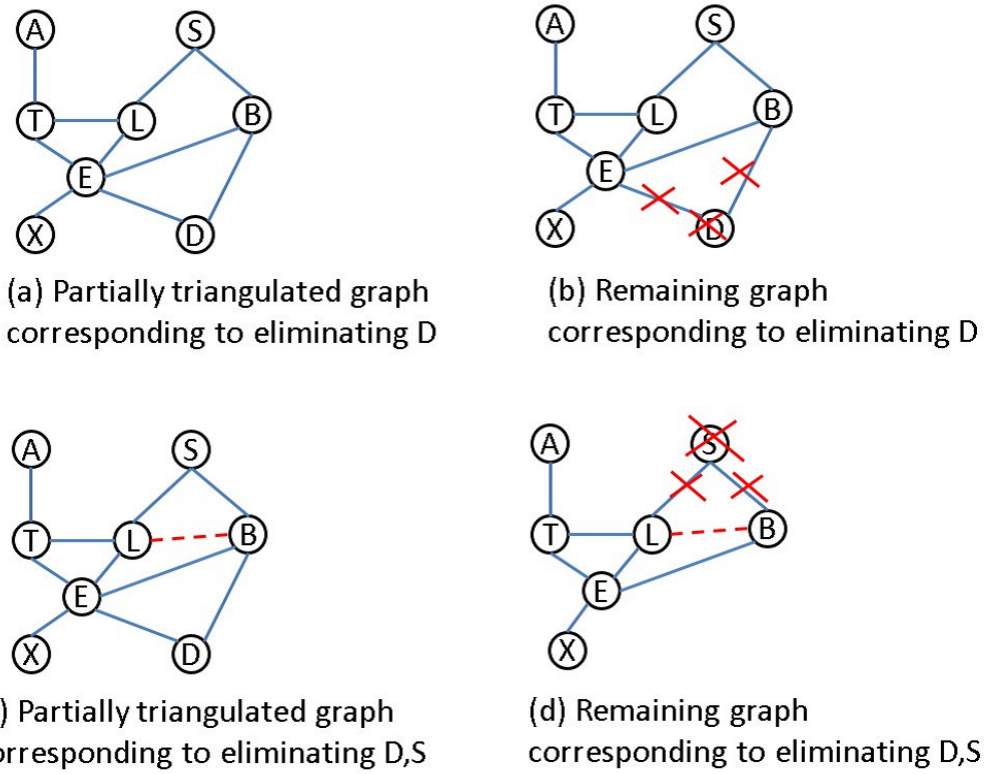


Figure 2.6: An example of eliminating vertices from the moral graph of the Asia network.

the final partially triangulated graph H^π is a chordal graph that has no chord-less cycles. Triangulation according to a particular elimination order is simple, but the determination of an optimal elimination order is the most important step. In this thesis, we try to find the order π for eliminating graph G that induces a chordal graph H^π with the minimum total table size.

2.2.2 Search space of the optimal triangulation algorithm

To find an optimal triangulation of a Bayesian network, we can conduct a search in the space of all elimination orders of the Bayesian network [Ottosen and Vomlel, 2012]. For this purpose, we generate a search graph that includes all elimination orders of the Bayesian network. The search graph is a tree with root node corresponding to the initial search node and leaf nodes corresponding to all distinct elimination orders. Figure 2.7 depicts the search space of the optimal triangulation algorithm on a network graph with five vertices. In this search tree, each non-leaf node is labeled using a partial elimination order τ that is a sequence of vertices for ordering the elimination process. We also associate the partially triangulated graph H^τ and the remaining graph G^τ with each node for reasons of computational convenience in the optimal triangulation algorithm. Each child of a node τ is generated by eliminating a vertex from its parent's remaining graph G^τ and appending that vertex to its parent's partial elimination order τ. By exploring the search tree, we can find an elimination order that induces a chordal graph with the minimum total table size.

2.3 Heuristic triangulation algorithms

In this section, we describe three common heuristic triangulation algorithms (minimum fill-in, minimum degree and minimum weight), which all greedily pick the next vertex to eliminate based on a local score [Kjaerulff, 1990; Wen, 1990; Dar-

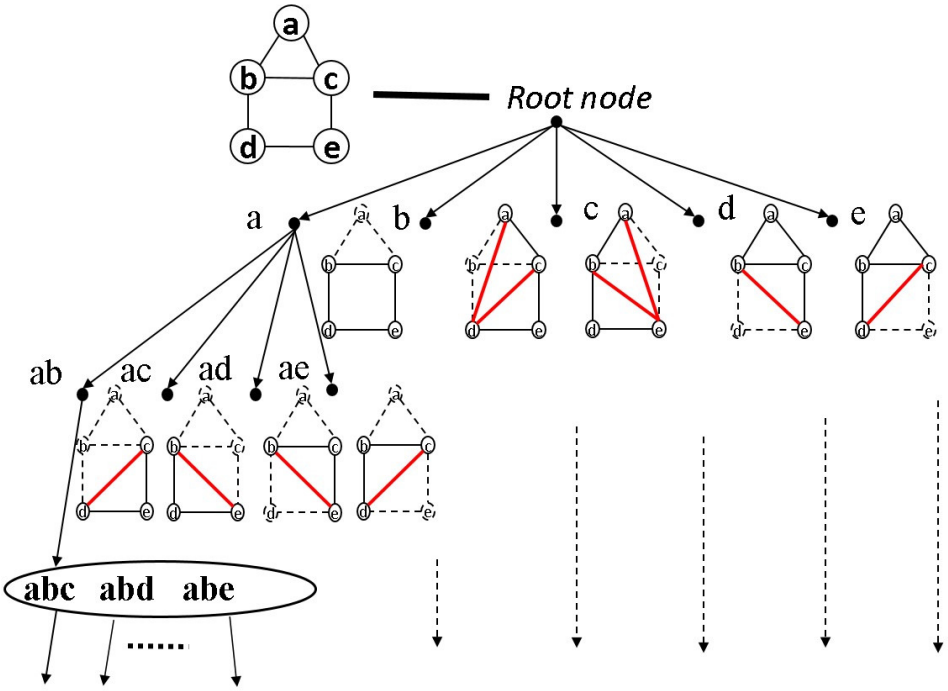


Figure 2.7: The search tree of the optimal triangulation algorithm for a network graph with five vertices.

wiche, 2009]. These heuristics have shown that they can generate a reasonably good upper bound on treewidth and, indirectly, can generate a small total table size triangulation. Although these approximation methods are not guaranteed to find a triangulation with minimum total table size, they generate an approximation in polynomial time with respect to the size of the graph and therefore can be used for computing an upper bound for optimal triangulation methods. The minimum fill-in algorithm greedily selects the next vertex to eliminate if the elimination adds the minimum number of fill-in edges; the minimum degree algorithm greedily selects the next vertex to eliminate if it has the minimum number of neighboring vertices; the minimum weight algorithm greedily selects the next vertex to eliminate if the product of weights of its neighbors is a minimum.

2.4 The optimal triangulation algorithm

This section reviews the depth-first search optimal triangulation algorithm presented by Ottosen and Vomlel [Ottosen and Vomlel, 2012]. The naive depth-first branch and bound algorithm for optimal triangulation operates as follows. First, the algorithm initializes the upper bound (UB) on total table size (tts) with the triangulation obtained by the minimum fill-in heuristic (MinFill). Next, it traverses all search tree nodes in a depth-first manner. For each search tree node, we calculate the tts of the partially triangulated graph corresponding to the node. This quantity is a lower bound for the tts of the node because the tts of a graph cannot be decreased by adding edges [Ottosen and Vomlel, 2012]. If we find a node for which the tts is greater than the tts of UB, then we prune all the descendants of the node. On the other hand, if we find a leaf node for which the tts is smaller than UB, we update UB by replacing UB with the leaf node (including the chordal graph and the tts of the node). The search continues until all nodes have been explored. At completion, the algorithm finds an optimal order or, equivalently, an optimal triangulation. It is noteworthy that the algorithm explores the search space of all elimination orders.

In the depth-first search algorithm, we intend to use the tts upper bound for pruning nodes that have a greater tts. Therefore, we need to compute the tts of each node in the search tree. The tts of a node is easy to compute if we know the cliques of the partially triangulated graph corresponding to the node. To compute the tts of a node time-efficiently, Ottosen and Vomlel associate the following with each node t [Ottosen and Vomlel, 2012].

- $t.\tau$: The ordered list of vertices representing the partial elimination order.

- $t.H$: The partially triangulated graph obtained by adding all fill-in edges accumulated along the τ to the original moral graph.

- $t.\mathcal{C}$: The set $\mathcal{C}(H)$ of cliques for H.
- $t.tts$: The total table size of graph H, which is a lower bound on the tts of node t.
- $t.R$: The remaining graph, $R = H[V \backslash \mathcal{V}(\tau)]$, where $\mathcal{V}(\tau)$ denotes the set of vertices that lie in τ.

To compute $t.tts$, we need to compute the set of cliques $t.\mathcal{C}$ first. For this purpose, we can use a standard clique enumeration algorithm, such as the well-known Bron–Kerbosch (BK) algorithm [Cazals and Karande, 2008]. Now, we present an example to explain the lower bound and related computations.

Example 1. *Figure 2.8 depicts the vertex elimination process according to the leftmost path in Figure 2.7. The path corresponds to the sequential elimination of vertices a and b. The root node r corresponds to the graph on the left in Figure 2.8 (initial graph), where no vertex has been eliminated. We can compute the cliques of the root node's graph* $r.\mathcal{C} = \{\{a,b,c\},\{b,d\},\{d,e\},\{c,e\}\}$ *using the BK algorithm. In this case, the TTS (assuming all variables are binary) is* $3 \cdot 2^2 + 2^3 = 20$*, which is a lower estimate of the optimal TTS.*

The successor node t of r (induced by elimination of vertex a) corresponds to the middle-left graph in Figure 2.8. The partially triangulated graph t.H is the same graph as the initial one. Therefore, we can derive $t.tts = 20$*.*

We generate the successor node t' *of t (corresponding to the elimination of vertex a and then vertex b). The induced partially triangulated graph* $t'.H$ *corresponds to the middle-right graph in Figure 2.8, which includes the fill-in edge* (c, d)*. Note that when we introduce fill-in edges in eliminating vertex b, we must not add edge* (a, d) *because the vertex a has been eliminated and is not present in the remaining graph, even though both a and d are neighbors of b in the partially triangulated graph. Since graph* $t'.H$ *is a chordal graph, it is not necessary*

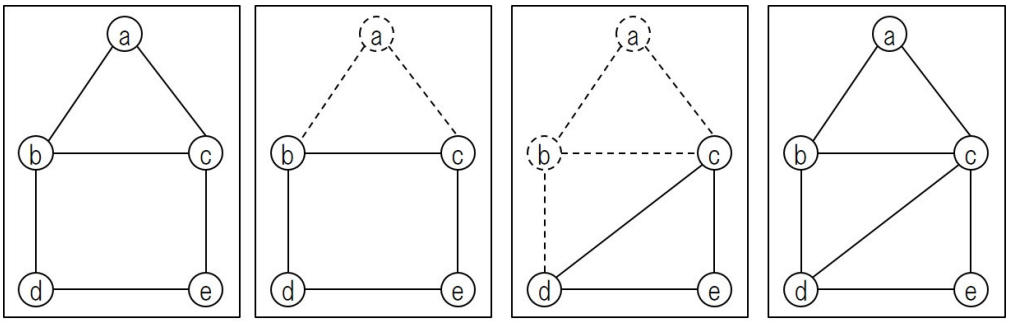


Figure 2.8: An example of the vertex elimination process according to the elimination order that starts with sequence $\langle a, b\rangle$. Left: Initial graph. Middle left: Partially triangulated graph corresponding to partial elimination order $\langle a\rangle$. Middle right: Partially triangulated graph corresponding to the partial elimination order $\langle a, b\rangle$. Right: Final chordal graph.

to generate any successor of t'. *Finally, the cliques of the chordal graph are* $t'.\mathcal{C} = \{\{a,b,c\},\{b,c,d\},\{c,d,e\}\}$ *and* $t'.tts$ *is* $3 \cdot 2^3 = 24$. *In this example, we can see that the TTS of a node is never higher than the TTS of its successor nodes. This key property ensures the correctness of applying the branch and bound technique in the optimal triangulation algorithm.*

Unfortunately, the BK algorithm has a heavy computational cost. Because eliminating one vertex changes only a small part of a partially triangulated graph, performing the BK algorithm on the whole graph results in many redundant computations. To tackle this problem, Ottosen and Vomlel [Ottosen and Vomlel, 2012] proposed a more efficient algorithm for computing the set of cliques in a dynamic graph. We will explain this dynamic clique maintenance algorithm in Section 3.2. However, the dynamic clique maintenance algorithm proposed by Ottosen and Vomlel allows computing some duplicate cliques. To resolve this problem, we propose a new dynamic clique maintenance algorithm in Chapter 3.

For a Bayesian network with n variables, the depth-first search algorithm presented by Ottosen and Vomlel can be implemented in $\mathcal{O}^*(2^n)$ space and $\mathcal{O}^*(n!)$

Algorithm 1 Depth-first search with coalescing and upper-bound pruning.

```
1: function TriangulationByDFS(G)
2:     Let s = (G, C(G), tts(G), V)
3:     EliminateSimplicial(s)                          ▷ Simplicial vertex rule
4:     if |V(s.R)|=0 then
5:         return s
6:     end if
7:     Let best = MinFill(s)                           ▷ Best path
8:     Let map = ∅                                     ▷ Coalescing map
9:     ExpandNode(s, best, map)                        ▷ Start recursive call return best
10: end function
11: procedure ExpandNode(t, &best, &map)
12:     for all v ∈ V(t.R) do
13:         Let m = Copy(t)
14:         EliminateVertex(m, v)                       ▷ Update graph, cliques and tts
15:         EliminateSimplicial(m)                      ▷ Simplicial vertex rule
16:         if |V(m.R)|=0 then
17:             if m.tts < best.tts then
18:                 Set best = m
19:             end if
20:         else
21:             if m.tts ≥ best.tts then
22:                 continue                            ▷ Branch and bound
23:             end if
24:             if map[m.R].tts ≤ m.tts then
25:                 continue
26:             end if
27:             Set map[m.R] = m
28:             ExpandNode(m, best, map)
29:         end if
30:     end for
31: end procedure
```

time. Pseudocode for the Ottosen and Vomlel algorithm is outlined in Algorithm 1. The procedure EliminateVertex(m, v) eliminates vertex v from the remaining graph of node m and simultaneously updates the set of cliques and the total table size. To prune unnecessary search nodes further, Ottosen and Vomlel also introduced the following pruning rules: (1) a graph reduction technique called the simplicial vertex rule [Bodlaender et al., 2005; van den Eijkhof et al., 2007], and (2) coalescing of nodes [Dow and Korf, 2007; Darwiche, 2009]. The procedure EliminateSimplicial(m, v) sequentially removes all simplicial vertices from the remaining graph of node m. The coalescing map uses $\mathcal{O}^*(2^n)$ space to prune unnecessary search nodes (see [Dow and Korf, 2007; Darwiche, 2009; Ottosen and Vomlel, 2012] for details). Although the algorithm combined with the above pruning techniques reduces the actual running time, it runs in $\mathcal{O}^*(n!)$ time in the worst case because it might explore the search space of all elimination orders, which has size $n!$. It is known that some different elimination orders induce identical triangulations. Consequently, the Ottosen and Vomlel algorithm might explore a large number of equivalent elimination orders. In Section 4.2, we propose a pruning rule, called pivot clique pruning, that removes a large number of superfluous elimination orders from the search space.

Chapter 3

Dynamic clique maintenance

3.1 Introduction

In this section, we consider the problem of maintaining a set of cliques of a dynamic undirected graph. A dynamic graph is defined as a graph in which the edges can be removed and added, but the set of vertices is invariant. The dynamic clique maintenance algorithm is a method to find all cliques of a dynamic graph. Let $G = (V, E)$ be a graph and $G' = (V, E \cup F)$ be the graph resulting by adding a set of new edges F to $G = (V, E)$. In particular, the dynamic clique maintenance algorithm computes the set of cliques in G' given that the set of cliques in G is already known.

This study is motivated by the optimal triangulation of Bayesian networks with respect to the total table size criterion using a best-first or depth-first search. This requires a lower bound at each node on the total table size, for which we use the total table size of the partially triangulated graph that is associated to the node. In turn, this requires that we know all the cliques of each partially triangulated graph in the triangulation process.

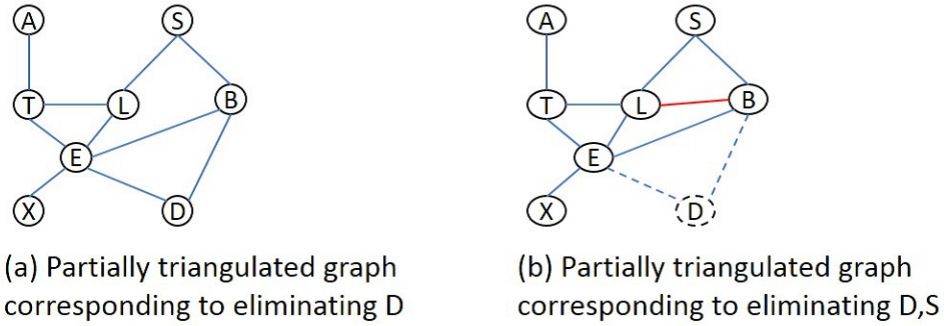
3.2 Previous work on dynamic clique maintenance

Algorithm 2 Dynamic clique maintenance algorithm proposed by Ottosen and Vomlel.

```
1:  procedure CliqueUpdate(G, C(G), F)
2:      Let G' = (V, E ∪ F)
3:      Let C(G') = C(G)
4:      Let U = V(F)
5:      for each clique C ∈ C(G') do                    ▷ Remove old cliques
6:          if C ∩ U ≠ ∅ then
7:              Set C(G') = C(G') \ {C}
8:          end if
9:      end for
10:     Let C^new = BKalgorithm(G'[FA(U, G')])
11:     for each clique C ∈ C^new do                    ▷ Add new cliques
12:         if C ∩ U ≠ ∅ then
13:             Set C(G') = C(G') ∪ {C}
14:         end if
15:     end for
16: end procedure
```

To avoid searching for all cliques in the whole graph as the BK algorithm does, Ottosen and Vomlel proposed a dynamic clique maintenance algorithm that runs a clique enumeration algorithm on a smaller subgraph in which all the new cliques can be found and all the existing cliques are removed [Ottosen and Vomlel, 2012]. This dynamic clique maintenance is presented in Algorithm 2, where G is the initial graph, $\mathcal{C}(G)$ is the set of cliques of G, F signifies the fill-in edges, and G' is derived by adding F to G. BKalgorithm(G) returns a set of cliques of the graph G. The dynamic clique maintenance algorithm is based on the following theorem.

Theorem 3.1 ([Ottosen and Vomlel, 2012])**.** *Let $G = (V, E)$ be an undirected*



(a) Partially triangulated graph corresponding to eliminating D

(b) Partially triangulated graph corresponding to eliminating D,S

Figure 3.1: A sequence of graphs corresponding to eliminating vertex D followed by vertex S in an order that starts with $\langle D, S\rangle$. (L, B) is the fill-in edge.

graph, and let $G' = (V, E \cup F)$ *be the graph resulting from adding a set of new edges* F *to* G*. Let* $U = \mathcal{V}(F)$*, and let* $\mathcal{C}(G')$ *be initialized with* $\mathcal{C}(G)$*. If the cliques in* G *that intersect with* U *are removed from* $\mathcal{C}(G')$ *and the cliques in* $G'[\mathcal{FA}(U, G')]$ *that intersect with* U *are added to* $\mathcal{C}(G')$*, then* $\mathcal{C}(G')$ *is the set of cliques of* G'*.*

Next, we provide an example to illustrate Algorithm 2.

Example 2. *Consider the Figure 3.1a.* $\mathcal{C}(G)$ *is the set of cliques of* G*,* $\{\{A, T\}$*,* $\{T, L, E\}$*,* $\{E, X\}$*,* $\{S, L\}$*,* $\{S, B\}$*,* $\{B, D, E\}\}$*. We add fill-in edges* $F = \{(L, B)\}$ *to graph* G*, resulting in new graph* G' *(corresponding to the graph in Figure 3.1b). The set* $U = \{L, B\}$*, and we let* $\mathcal{C}(G')$ *be initialized with* $\mathcal{C}(G)$*.*

First, we scan through the cliques in $\mathcal{C}(G')$ *to remove the cliques that intersect with* U*, which is the set of cliques* $\{\{T, L, E\}$*,*$\{S, L\}$*,*$\{S, B\}$*,*$\{B, D, E\}\}$*. Next, we run the BK algorithm on a subgraph* $G'[\mathcal{FA}(U, G')]$*. Thereby, we obtain* C^{new}*=*$\{\{T, L, E\}$*,*$\{S, L, B\}$*,*$\{L, B, E\}$*,*$\{B, D, E\}\}$*.*

Finally, we add to $\mathcal{C}(G')$ *all the cliques found in the subgraph* $G'[\mathcal{FA}(U, G')]$ *that intersect with* U*. Now* $\mathcal{C}(G')$*=*$\{\{A, T\}$*,*$\{E, X\}$*,*$\{T, L, E\}$*,*$\{S, L, B\}$*,* $\{L, B, E\}$*,* $\{B, D, E\}\}$*, which is the set of cliques of the new graph* G'*.*

The example shows that the algorithm sometimes removes a clique and then adds it again. Although the Ottosen and Vomlel method reduces the search space

of the BK algorithm from the whole graph G' to a small subgraph $G'[\mathcal{FA}(U, G')]$, the method might present shortcomings in performance when the number of duplicate cliques becomes large. In this example, we observed that vertex D has been eliminated. It is well known that a new fill-in edge cannot connect to a vertex that has been eliminated. Because the neighbors of D are invariant in G and G', any clique containing D in the initial graph should remain a clique in the updated graph. Generally, no clique containing one of the eliminated vertices should be calculated again. Based on this observation, Li and Ueno [Li and Ueno, 2012] proposed an improved dynamic clique maintenance algorithm. The Li and Ueno dynamic clique maintenance procedure is shown in Algorithm 3, where $G, \mathcal{C}(G), F$ are defined in the same manner as presented in Algorithm 2, and W is the set of vertices that have been eliminated before. The improved dynamic clique maintenance runs the BK algorithm on the graph $G'[\mathcal{FA}(U, G') \setminus W]$, which is a subgraph of $G'[\mathcal{FA}(U, G')]$ that the Ottosen and Vomlel method explores. Because the complexity of the BK algorithm is exponential in the number of vertices in the subgraph, reducing the search space of the BK algorithm is important for improving the performance of dynamic clique maintenance. In the Li and Ueno method, when we remove an old clique C, one more conditional check is necessary to ascertain whether clique C and W are disjoint. This check is usually not a problem because the complexity of comparison of two cliques is constant if we store a clique using a BitSet Object in the JAVA programming language. However, the method still computes many duplicate cliques.

Algorithm 3 Dynamic clique maintenance algorithm proposed by Li and Ueno (2012).

procedure CLIQUEUPDATE1($G, \mathcal{C}(G), F, W$)
  Let $G' = (V, E \cup F)$
  Let $\mathcal{C}(G') = \mathcal{C}(G)$
  Let $U = \mathcal{V}(F)$
  for each clique $C \in \mathcal{C}(G')$ **do** ▷ Remove old cliques
    if $C \cap U \neq \emptyset$ **then**
      if $C \cap W = \emptyset$ **then**
        Set $\mathcal{C}(G') = \mathcal{C}(G') \setminus \{C\}$
      end if
    end if
  end for
  Let $C^{new} = \text{BKalgorithm}(G'[\mathcal{FA}(U, G') \setminus W])$
  for each clique $C \in C^{new}$ **do** ▷ Add new cliques
    if $C \cap U \neq \emptyset$ **then**
      Set $\mathcal{C}(G') = \mathcal{C}(G') \cup \{C\}$
    end if
  end for
end procedure

Algorithm 4 Proposed dynamic clique maintenance algorithm.

```
1: procedure CliqueUpdate2(G, C(G), F)
2:     Let G' = (V, E ∪ F)
3:     Let C(G') = C(G)
4:     Let W = FA(F, G')
5:     for each clique C ∈ C(G') do                    ▷ Remove old cliques
6:         if C ⊆ W then
7:             Set C(G') = C(G') \ {C}
8:         end if
9:     end for
10:    C(G') = C(G') ∪ BKalgorithm(G'[W])              ▷ Add new cliques
11: end procedure
```

3.3 Proposed dynamic clique maintenance algorithm

In the depth-first search optimal triangulation algorithm, it is necessary to compute the lower bound of tts for each search node. Therefore, the computational cost of updating cliques is inherent in expanding each node. To lower the overhead cost of each node, we must compute the cliques of each graph efficiently. In Section 3.2, we have demonstrated by example that the Ottosen and Vomlel approach might compute some duplicate cliques. To resolve this problem, we propose a new dynamic clique maintenance algorithm. When some new edges are inserted into a graph, a new clique contains at least one new edge. The main idea of our method is to avoid recomputing the cliques that do not contain a new edge.

For a graph $G = (V, E)$ and an edge $e = (v, w) \in E$, we define the neighborhood $\mathcal{N}(e, G)$ of an edge e as the set of vertices $U \subseteq V$ such that U contains all the vertices adjacent to both v and w. For a set of edges F, the family $\mathcal{FA}(F, G)$ of F is defined as the set $(\cup_{f \in F} \mathcal{N}(f, G)) \cup \mathcal{V}(F)$. Let $G = (V, E)$ be the initial graph

and let $G' = (V, E \cup F)(F \cap E = \emptyset)$ be the graph obtained by adding a set of new edges F to G. All new cliques and removed cliques are included in the vertex set $\mathcal{FA}(F, G')$ according to the following theorem. Therefore, we can run the BK algorithm on only the subgraph $G[\mathcal{FA}(F, G')]$. Note that the family $\mathcal{FA}(F, G')$ is a subset of the family $\mathcal{FA}(\mathcal{V}(F), G')$, which is the subgraph explored during the Ottosen and Vomlel method. The proposed dynamic clique maintenance is shown in Algorithm 4, where $G, F, \mathcal{C}(G)$ are defined in the same manner as presented in Algorithm 2, and $W = \mathcal{FA}(F, G')$ denotes the family of a set of edges F.

The new algorithm is based on the following theorem.

Theorem 3.2. *Let $G = (V, E)$ be an undirected graph, and let $G' = (V, E \cup F)$ be the graph resulting from adding a set of new edges F to G. Let $W = \mathcal{FA}(F, G')$, and let $\mathcal{C}(G')$ be initialized with $\mathcal{C}(G)$. If the cliques that are included in W are removed from $\mathcal{C}(G')$ and the cliques of $\mathcal{C}(G'[W])$ are added to $\mathcal{C}(G')$, then $\mathcal{C}(G')$ is the set of cliques of G'.*

Proof. If C is a complete set in $\mathcal{NC}(G, G')$ (which means $C \in \mathcal{C}(G')$ and $C \notin \mathcal{C}(G)$), then C contains at least one new edge $f \in F$; otherwise C would be a clique in G. If C is a new clique that contains a new edge $f = (v, w) \in F$, then any vertex $u \in C$ ($u \neq v$ or w) is included in $\mathcal{N}(f, G')$. Therefore, $C \subseteq \mathcal{FA}(F, G')$. Thus, all the new cliques can be found in the subgraph $G[\mathcal{FA}(F, G')]$.

If K is a complete set in $\mathcal{RC}(G, G')$ (which means $K \in \mathcal{C}(G)$ and $K \notin \mathcal{C}(G')$), then there exists a new clique C such that $K \subseteq C$. Because any new clique is included in $\mathcal{FA}(F, G')$ as proved above, $C \subseteq \mathcal{FA}(F, G')$. Therefore, each removed clique K is included in $\mathcal{FA}(F, G')$.

We remove all the old cliques by removing all the cliques included in $\mathcal{FA}(F, G')$ from $\mathcal{C}(G')$, and then add all the new cliques which can be found in the subgraph G$[\mathcal{FA}(F, G')]$ to $\mathcal{C}(G')$. Then, $\mathcal{C}(G')$ is the set of cliques of G'. □

The following example illustrates the algorithm.

Example 3. *Consider again the graph G and updated graph G' in Figure 3.1. $\mathcal{C}(G)$ is the set of cliques of G, $\mathcal{C}(G) = \{\{A,T\},\{T,L,E\},\{E,X\},\{S,L\},\{S,B\},\{B,D,E\}\}$. Let $\mathcal{C}(G')$ be initialized with $\mathcal{C}(G)$. We first compute the family of edge set F, $W = \mathcal{FA}(F,G') = \{S,E,L,B\}$. Next, we scan through the cliques in $\mathcal{C}(G')$ to remove all the cliques included in W, which is the set of cliques $\{\{S,L\},\{S,B\}\}$.*

Then, we run the BK algorithm on a subgraph $G'[W]$. We obtain $C^{new} = \{\{S,L,B\},\{L,B,E\}\}$. In the Ottosen and Vomlel method, we run the BK algorithm on $G'[\mathcal{FA}(U,G')]$, where $\mathcal{FA}(U,G') = \{S,T,E,D,L,B\}$. However, in our new method, we run the BK algorithm on $G'[W]$, where $W = \{S,E,L,B\}$. It can be easily proved that vertex set $W = \mathcal{FA}(F,G')$ is always a subset of $\mathcal{FA}(\mathcal{V}(F),G')$. Our method makes the BK algorithm explore less search space for updating cliques than the Ottosen and Vomlel method. Since the complexity of the BK algorithm is exponential in the number of vertices in the subgraph, this reduction is important to improve the performance of dynamic clique maintenance.

Finally, we simply add all new cliques C^{new} to $\mathcal{C}(G')$. In this example, we only remove cliques $\mathcal{RC}(G,G')$ from $\mathcal{C}(G')$ and add cliques $\mathcal{NC}(G,G')$ to $\mathcal{C}(G')$. In contrast, the Ottosen and Vomlel method removes some duplicate cliques and then adds them again.

Given a graph G, a set of new edges F and the eliminated vertex set W, consider the problem of computing the set of cliques of new graph G'. The Ottosen and Vomlel method runs the BK algorithm on $G'[\mathcal{FA}(\mathcal{V}(F),G')]$, the Li and Ueno method runs the BK algorithm on $G'[\mathcal{FA}(\mathcal{V}(F),G')\setminus W]$ and the proposed method runs the BK algorithm on $G'[\mathcal{FA}(F,G')]$. The BK algorithm suffers from heavy computational cost, and the proposed method reduces the search space of the BK algorithm because $\mathcal{FA}(\mathcal{V}(F),G') \supseteq \mathcal{FA}(\mathcal{V}(F),G') \setminus W \supseteq \mathcal{FA}(F,G')$. Therefore, our proposed method is expected to dramatically reduce the running time of dynamic clique maintenance. In the Ottosen and Vomlel approach, it is necessary to check each clique in $G'[\mathcal{FA}(\mathcal{V}(F),G')]$ to ascertain whether it intersects $\mathcal{V}(F)$.

However, we can remove this conditional check from our algorithm.

The dynamic clique maintenance algorithms has two main steps: scanning all existing cliques and running the BK algorithm. If dynamic clique maintenance algorithms are used for computing graphs with too many cliques, then the scanning part will dominate the complexity of the dynamic clique maintenance because they need to scan all existing cliques in the graphs. In this case, the three dynamic clique maintenance algorithms are expected to perform equally well. Fortunately, our study of the repository of Bayesian networks with less than 100 vertices found that there are not so many cliques in these network graphs. Except on graphs with many cliques, our proposed algorithm is expected to run faster than the Ottosen and Vomlel method, because it reduces the search space of the BK algorithm. We demonstrate the superior performance of the new algorithm by considering the results of simulation experiments in the next section.

3.4 Experiments

We conducted computational experiments to evaluate our proposed algorithms on a set of benchmark Bayesian networks. These networks are obtained from the well-known Bayesian Network Repository [Scutari, 2016]. We also generated a set of random graphs for doing controlled experiments. A random graph G is generated by successively adding random edges to a set of vertices. Here, a random edge was generated by picking its two endpoints uniformly at random from all unconnected pairs of vertices. We compared our algorithm with state-of-the-art algorithms on the benchmark networks and the random graphs. All the algorithms described in this thesis are implemented in the Java language, and the source code is available at http://www.ai.lab.uec.ac.jp/optimaltriangulation/. The experiments were performed on a Windows 10 PC with a 2.6 GHz Intel Xeon Processor E5-2640 and 16GB RAM, running version 8 of the Java Virtual Machine.

3.4.1 Depth-first search with proposed dynamic clique maintenance

We first evaluated the computational costs of our proposed dynamic clique maintenance algorithm and state-of-the-art algorithms. In particular, for each algorithm, we compared the total number of cliques that have to be enumerated for optimal triangulation of the repository Bayesian networks. For this purpose, we implemented the following algorithms.

- DFS (OandV): the depth-first search (DFS) optimal triangulation algorithm proposed by Ottosen and Vomlel, which uses the Ottosen and Vomlel approach [Ottosen and Vomlel, 2012] for dynamic clique maintenance.
- DFS (LandU2012): the DFS algorithm with the Li and Ueno approach for dynamic clique maintenance [Li and Ueno, 2012].
- DFS (Proposed): the DFS algorithm with the proposed method of dynamic clique maintenance.

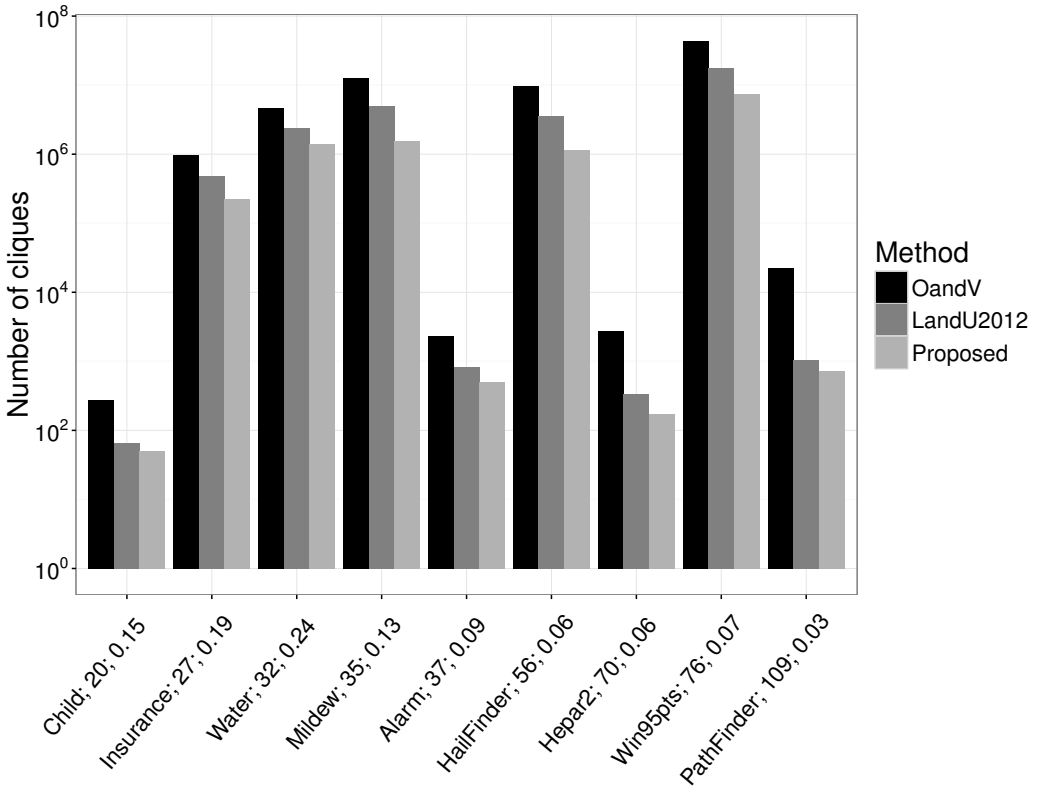


Figure 3.2: A comparison of the number of cliques that are enumerated by each algorithm. Each label on the X-axis consists of the network name, the number of variables, and the graph density for a Bayesian network.

Figure 3.2 shows the number of cliques enumerated by each algorithm. For the Barley Bayesian network in the repository, no triangulation algorithm completes the computation within one hour. Therefore, we report the results for only nine Bayesian networks. The number of cliques enumerated by our proposed method is lower than the number of any other method. The reason is that the proposed method reduces the number of duplicate cliques by reducing the search space of the BK algorithm in dynamic clique maintenance. As described in Section 3.2, the Ottosen and Vomlel method explores a subgraph with many cliques that do not contain new edges. In contrast, our proposed algorithm explores an even smaller subgraph that contains only the vertices connected by new edges and all neighboring vertices of new edges. Consequently, the proposed algorithm reduces the search space of the BK algorithm in dynamic clique maintenance and excludes a large number of duplicate cliques.

To verify the superior performance of our proposed algorithm, we compared the running times of DFS (OandV), DFS (LandU2012) and DFS (Proposed). Note that the three algorithms explore the exact same search space and thus the differences between the computational times of the algorithms are only caused by differences of running times for dynamic clique maintenance. We also computed the ratio of running time for each algorithm to the running time of the DFS (OandV) algorithm. Table 3.1 shows the computational time and the time ratio for each algorithm.

Our proposed dynamic clique maintenance remarkably improves the running time of optimal triangulation. The reason is that the BK algorithm has a heavy computational cost and the proposed method reduces the search space of the BK algorithm. For Hepar2 and PathFinder, respectively, DFS (Proposed) is 12.04 and 6.5 times as fast as the DFS (OandV) method. From Figure 3.2, we can see that the reason for this is that DFS (Proposed) computes much fewer cliques than DFS (OandV) does on the two networks.

Table 3.1: A comparison of the running times (s) for the Ottosen and Vomlel method (OandV), the Li and Ueno method (LandU2012) and the Proposed method.

Bayesian Networks				OandV		LandU2012		Proposed	
Names	V	E	Density	Time	OandV/OandV	Time	OandV/LandU2012	Time	OandV/Proposed
Child	20	30	0.15	0.0053	1	0.0014	3.79	0.0009	5.89
Insurance	27	70	0.19	1.81	1	1.07	1.69	0.68	2.66
Water	32	123	0.24	9.86	1	5.96	1.65	4.52	2.18
Mildew	35	80	0.13	18.26	1	7.16	2.55	4.45	4.1
Alarm	37	65	0.09	0.0192	1	0.0054	3.56	0.0042	4.57
HailFinder	56	99	0.06	13.13	1	5.46	2.4	3.8	3.46
Hepar2	70	158	0.06	0.0289	1	0.003	9.63	0.0024	12.04
Win95pts	76	225	0.07	94.42	1	38.89	2.43	26.86	3.52
PathFinder	109	208	0.03	0.0637	1	0.0121	5.26	0.0098	6.5

3.4.2 Dynamic clique maintenance on random graphs

The comparison of dynamic clique maintenance algorithms on random graphs was done as follows. We generated 40 random graphs with various densities for each of 25, 50 and 75 vertices. Then we performed the following test on the dataset. We triangulated each graph in the dataset 1,000 times by sequentially eliminating all vertices (with a different random elimination order on each run) and saved the total running time. The set of cliques of the graph was updated after each vertex was eliminated. We then normalized these times to ensure a fair comparison among graphs with different sizes. For example, the task of triangulating a graph with 25 vertices 1,000 times performed 25,000 dynamic clique maintenance steps. Therefore, we normalized this time by dividing by 25. Figure 3.3 depicts the normalized running times of 1,000 triangulations of each graph in the dataset. The results show that the proposed dynamic clique maintenance algorithm is faster than both the OandV method and the LandU2012 method for all the random graphs except four data sets (random graphs with more than 50 vertices and density of greater than 0.3). The reason for the difference on these data sets is that there are too many cliques in a dense graph and so scanning for cliques dominates the computational complexity. For the ten graphs with 75 vertices and a density of 0.1, the average number of cliques is 56.3. Because running the BK algorithm dominates the complexity in this case, our proposed algorithm is the fastest algorithm. However, for the ten graphs with 75 vertices and a density of 0.4, the average number of cliques is 986.8. Because the scanning for cliques dominates the computational complexity in this case and the three algorithms have to scan through equally many cliques, the three algorithms performed almost equally well. Because the maximum number of cliques in a graph with n vertices and maximum degree d is bounded by $\mathcal{O}(n \cdot 2^d)$ [Wood, 2007], we suspect that dense and large graphs tend to have more cliques than sparse graphs and so make the dynamic clique maintenance problem more difficult. Finally, it is noteworthy that a Bayesian network with a sparse

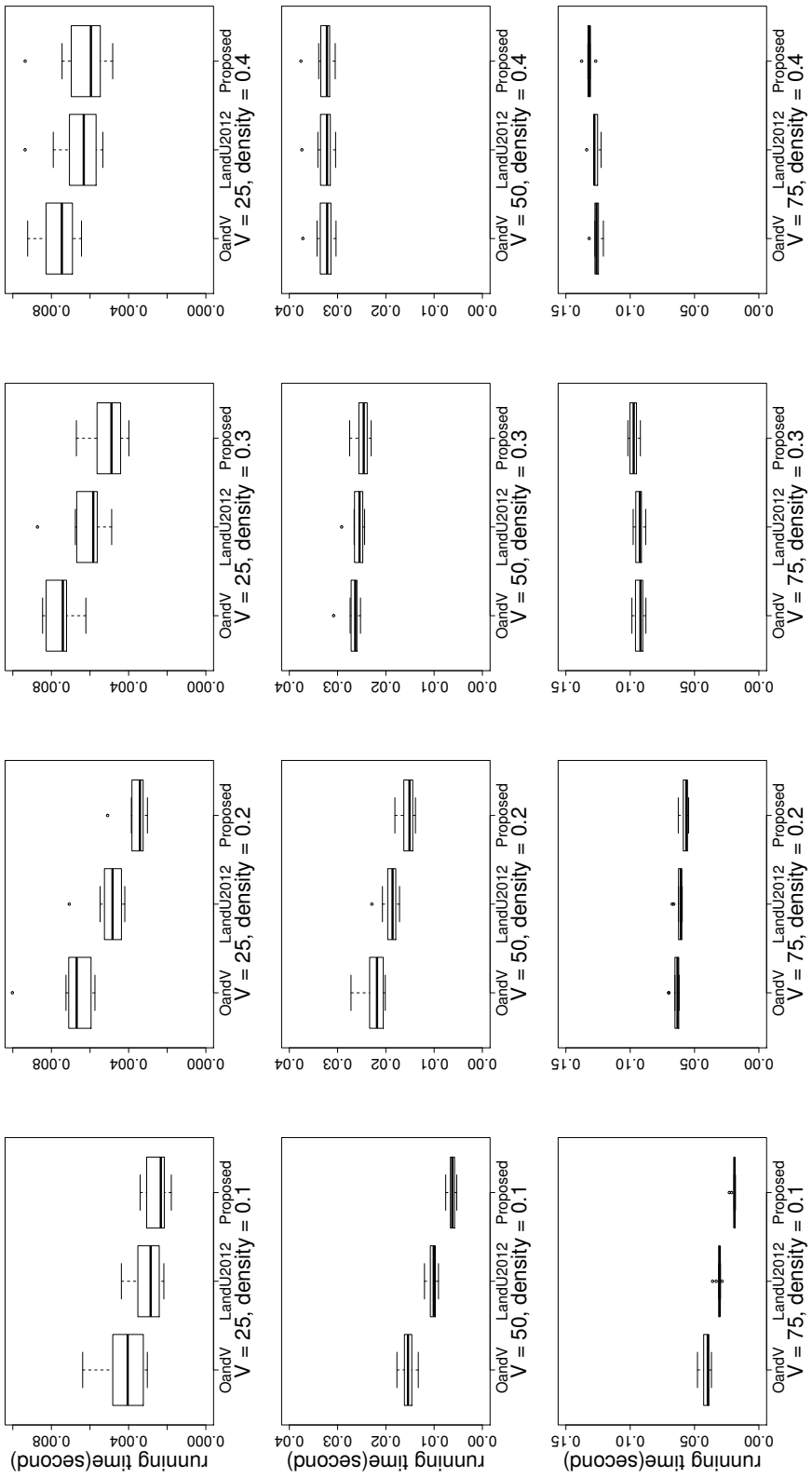


Figure 3.3: A comparison of the running times for different dynamic clique maintenance algorithms on the random graphs.

and small graph does not indicate an easy triangulation problem. For the Barley network with 48 vertices and a density of 0.11, none of the three DFS algorithms can find an optimal triangulation within the time limit.

Chapter 4

Pivot clique pruning

4.1 Introduction

In the worst case, the depth-first search algorithm described in Chapter 2 explores the search space of all elimination orders which has size $n!$, where n is the number of variables in the Bayesian network. It is known that different elimination orders can induce identical triangulations. Consequently, the depth-first search algorithm might unnecessarily explore a large number of elimination orders. In this chapter, we propose a novel pruning rule called pivot clique pruning, which can remove a large number of redundant elimination orders from the search space.

We first show an example of two different elimination orders leading to duplicate results. Consider the process of eliminating vertices from the left graph G in Figure 2.5. Let $\pi = \langle a, b, e, c, d \rangle$ be an elimination order. By exchanging the positions of b and e, we obtain the order $\pi' = \langle a, e, b, c, d \rangle$. If we eliminate all vertices from the graph G according to order π, then we obtain a chordal graph H^{π}, which is shown as the right graph in Figure 2.5. An important observation here is that order π' induces an identical chordal graph, that is, $H^{\pi'} = H^{\pi}$. Let G_1^{π} and $G_1^{\pi'}$ be initialized with graph G. We first eliminate vertex $\pi(1) = a$ from graph G_1^{π}, and vertex $\pi'(1) = a$ from graph $G_1^{\pi'}$, then we obtain the identical remaining graphs

$G_2^\pi = G_2^{\pi'}$. Next, we eliminate vertex $\pi(2) = b$ from graph G_2^π. In the elimination process, we add fill-in edge (c, d) to make the neighbors of vertex b, $\mathcal{N}(b, G_2^\pi)$ complete, and then we obtain the remaining graph G_3^π. Because the vertices b and e are not adjacent, the neighbors of e are the same in graphs G_2^π and G_3^π: that is, $\mathcal{N}(e, G_2^\pi) = \mathcal{N}(e, G_3^\pi)$. In contrast, consider eliminating vertex $\pi'(2) = e$ from graph $G_2^{\pi'}$. Doing so, we obtain the remaining graph $G_3^{\pi'}$. Because the vertices b and e are not adjacent, we also have the result $\mathcal{N}(b, G_2^{\pi'}) = \mathcal{N}(b, G_3^{\pi'})$. In the order π, the elimination of vertices b and then e makes $\mathcal{N}(b, G_2^\pi)$ and $\mathcal{N}(e, G_3^\pi)$ complete. In the order π', the elimination of vertices e and then b makes $\mathcal{N}(e, G_3^{\pi'})$ and $\mathcal{N}(b, G_3^{\pi'})$ complete. Note that $\mathcal{N}(b, G_2^\pi) = \mathcal{N}(b, G_3^{\pi'})$ and $\mathcal{N}(e, G_3^\pi) = \mathcal{N}(e, G_3^{\pi'})$, making these two identical sets complete requires identical fill-in edges. In addition, it is clear that the remaining graphs are also equivalent: $G_4^\pi = G_4^{\pi'}$ and $G_5^\pi = G_5^{\pi'}$. Thus, the fill-in edges obtained from the orders π and π' are identical.

4.2 Pivot clique pruning

In the optimal triangulation algorithm, if we know two orders engender identical triangulations, then we can prune one of the two orders from the search space. However, we need not to explicitly identify the equivalent orders. The following theorems offer a straightforward approach to prune redundant orders with extremely low computational cost.

Lemma 4.1. *Let G be a graph, and let $\pi = (v_1, \ldots, v_n)$ be an elimination order. The elimination of vertices from graph G according to order π induces a graph sequence $G_1^\pi, G_2^\pi, \ldots, G_n^\pi$, where $G_1^\pi = G$ and G_{i+1}^π is obtained by eliminating vertex v_i from graph G_i^π. Suppose there exist two vertices v_i and v_k $(i < k)$ such that v_k is nonadjacent to v_{k-1} in G_{k-1}^π and v_l is adjacent to v_{l+1} for $l = i, ..., k-2$, then by moving v_k directly before v_i to obtain a new order $\pi' = (v_1, \ldots, v_{i-1}, v_k, v_i, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$, the orders π and π' engender identical*

chordal graphs.

Proof. First, we prove for $l = i, ..., k-1$, v_l is nonadjacent to v_k. We prove this by contradiction. Assume that there exists a vertex $v_l, l \in [i, k-1]$ such that v_l is adjacent to v_k. Then, eliminating vertex v_l makes v_{l+1} adjacent to v_k, because v_l is adjacent to both v_{l+1} and v_k. Under this assumption, if we eliminated vertices sequentially from v_i to v_{k-2}, we would obtain the result that v_{k-1} is adjacent to v_k, which is a contradiction.

Next, we prove that the filled-in graphs satisfy $H^\pi = H^{\pi'}$. When a vertex v_l is eliminated, if a pair of neighbors of v_l is not linked, a fill-in edge is added between these two vertices. In the case of π', eliminating v_k before v_l $(l \in [i, k-1])$ does not add new neighbors to v_l, because v_k is nonadjacent to v_l. Note that the neighbors of v_k are also invariant. As a result, the fill-in edges introduced by eliminating v_l $(l \in [i, k])$ are invariant in the two orders π and π'. Thus, we obtain the result that $H^{\pi'} = H^\pi$. □

Lemma 4.2. *Let G be a graph, and let $\pi = (v_1, \ldots, v_n)$ be an elimination order. The elimination of vertices from graph G according to order π induces a graph sequence $G_1^\pi, G_2^\pi, \ldots, G_n^\pi$, where $G_1^\pi = G$ and G_{i+1}^π is obtained by eliminating vertex v_i from G_i^π. Suppose G_i^π is not complete, and let v_j $(i < j)$ be a vertex that is not adjacent to v_i. Then, by moving v_j directly before v_i, we obtain an order π'=(v_1, ..., v_{i-1}, v_j, v_i, ..., v_{j-1}, v_{j+1},..., v_n) with a tts that is smaller than or equal to the tts of π.*

Proof. The sequence of vertices (v_i, ... , v_n) is such that either

1. there exists a vertex v_k $(i < k)$ such that v_{k-1} is nonadjacent to v_k and v_l is adjacent to v_{l+1} for $l = i, ..., k-2$, or

2. v_l is adjacent to v_{l+1}, for $l = i, ..., n-1$.

First, we prove the theorem in the first case. Note that v_i and v_k are not adjacent, which was shown in the proof of Lemma 4.1. By moving v_k directly before v_i, we obtain the new order $\pi' = (v_1, \ldots, v_{i-1}, v_k, v_i, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$. Then, from Lemma 4.1, H^{π}=$H^{\pi'}$, and the tts of π is equal to that of π' from the definition of tts.

Next, we prove the theorem in the second case. Lemma 4.1 cannot be directly applied to prove the theorem, because v_l is adjacent to v_{l+1} for $l = i, ..., n-1$. Therefore, we first introduce a new order ω so as to use Lemma 4.1. Because G_i^{π} is not complete and G_{n-1}^{π} is complete, there necessarily exists a vertex v_m ($i \leq m < n-1$) such that G_m^{π} is not complete and G_{m+1}^{π} is complete. Either

(a) v_m is adjacent to all vertices in G_{m+1}^{π}, or

(b) there exists a vertex v_k in G_{m+1}^{π}, such that v_m is not adjacent to v_k.

We consider each of these cases in turn.

(a) In G_{m-1}^{π}, there exists a vertex v_k ($m < k$) that is not adjacent to v_{m-1}; otherwise, eliminating v_{m-1} would make G_m complete. By moving v_k directly before v_m, we obtain order ω=(v_1, …, v_{m-1}, v_k, v_m, …, v_{k-1}, v_{k+1},…, v_n). Because v_m is adjacent to all vertices in G_{m+1}^{π}, eliminating vertex v_m adds all possible fill-in edges to make G_m^{π} a complete graph. Therefore, order ω will not add different edges from order π. Thus, the filled-in graph H^{ω} is a subgraph of H^{π}. In this case, the tts of H^{ω} is smaller than or equal to the tts of H^{π} because the tts of a graph is greater than or equal to that of a subgraph [Ottosen and Vomlel, 2012].

Because v_{m-1} and v_k are not adjacent and v_l is adjacent to v_{l+1} for $l = i, ..., m-2$, v_i and v_k are not adjacent, which was shown in the proof of Lemma 4.1. For the order ω, by moving v_k directly before v_i, we obtain order $\pi' = (v_1, \ldots, v_{i-1}, v_k, v_i, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$. Since vertex v_k is not adjacent to v_{m-1}, and v_l is adjacent to v_{l+1} for $l = i, ..., m-2$, from Lemma 4.1, $H^{\omega} = H^{\pi'}$. Therefore, the tts of $H^{\pi'}$ is smaller than or equal to the tts of H^{π}.

(b) In the case of π, eliminating a vertex after vertex v_m does not introduce fill-in edges, because G^{π}_{m+1} is complete. By moving v_k directly before v_{m+1}, we obtain order ω=(v_1, ..., v_m, v_k, v_{m+1}, ..., v_{k-1}, v_{k+1},..., v_n). In the case of ω, eliminating a vertex after vertex v_m also does not introduce fill-in edges. Because the two orders π and ω introduce the same fill-in edges, $H^{\omega} = H^{\pi}$.

Because v_m and v_k are not adjacent and v_l is adjacent to v_{l+1} for $l = i, ..., m-1$, v_i and v_k are not adjacent, which was shown in the proof of Lemma 4.1. For the order ω, by moving v_k directly before v_i, we obtain order $\pi' = (v_1, \ldots, v_{i-1}, v_k, v_i, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$. The vertices v_m and v_k are nonadjacent, and v_l is adjacent to v_{l+1} for $l = i, ..., m-1$. From Lemma 4.1, $H^{\pi'} = H^{\omega}$. Therefore, the tts of $H^{\pi'}$ is equal to the tts of H^{π}. □

Now, using Lemma 4.2, the following pivot clique pruning theorem can be derived.

Theorem 4.1 (pivot clique pruning)**.** *Let G be the graph being triangulated, and let $t = (\tau, G^{\tau}, H^{\tau}, \mathcal{C}(H^{\tau}), tts(H^{\tau}))$ be a non-leaf node in the search tree, where $t.G^{\tau}$ is an incomplete graph. Pick an arbitrary clique in $\mathcal{C}(t.G^{\tau})$ as the pivot clique C_{pivot}. If a child node of t is derived by eliminating a vertex in C_{pivot}, then the child node and all its descendants can be pruned.*

Proof. The search tree branches on node t to generate a child node for each vertex v in the remaining graph $t.G^{\tau}$. Let U be the set of child nodes of t if the child is derived by eliminating a vertex in C_{pivot}, as shown in Figure 4.1. Let W be the set of all child nodes of t except U. We show the following sufficient condition to prove the theorem. For any leaf node x that is reachable from one node in U, there is another leaf node y that is reachable from one node in W, such that the tts of y is smaller than or equal to the tts of x.

Let x be an arbitrary leaf node that is reachable from a node t_A in U, where t_A is a child node of t derived by eliminating vertex A from $t.G^{\tau}$. Because t_A

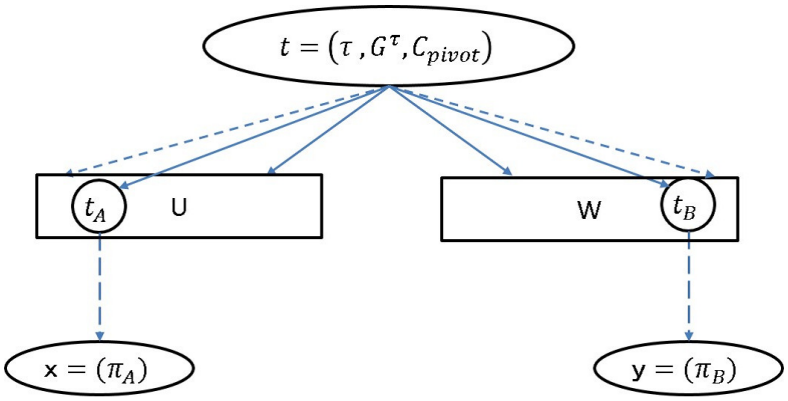


Figure 4.1: The part of search tree beginning at node t.

is in U, A is a vertex of C_{pivot}. The elimination order of node x is a complete elimination order π_A that is an extension of the partial elimination order $t_A.\tau$. Based on Lemma 4.2, there exists a vertex $B \in \mathcal{V}(t.G^{\tau})$ that is not adjacent to A, such that by moving vertex B directly before A in the order π_A, a new order π_B is obtained for which the tts is smaller than or equal to that of π_A. Let t_B be the child node of t derived by eliminating vertex B from $t.G^{\tau}$. Then the leaf node y labeled by π_B is reachable from a node t_B. Because B and A are not adjacent, A and B cannot be in the same clique. Since A is a vertex of clique C_{pivot}, B is not in C_{pivot}. Thus, t_B is in W. □

This theorem can be directly applied to prune some nodes in the search tree. Although pivot clique pruning might remove some optimal solutions, the reduced search tree is guaranteed to contain at least one optimal solution. The proposed depth-first search algorithm with pivot clique pruning is described in the Algorithm 5. The original depth-first search algorithm branches on a non-leaf node t for all the vertices in $\mathcal{V}(t.R)$, where $t.R$ is the remaining graph of node t. However, in our proposed algorithm on line 3, we generate only child nodes for the vertices in $\mathcal{V}(t.R) \setminus$ SelectPivotClique($\mathcal{C}(t.R)$). The procedure SelectPivotClique($\mathcal{C}(G)$) simply iterates through all the cliques of graph G to choose the largest clique of G.

We use this heuristic because it greedily prunes the largest number of child nodes. Finding the largest clique of the remaining graph $t.R$ seems to be expensive. However, it can be easily computed by finding the clique in $t.\mathcal{C}(H)$ such that the clique has the largest intersection with $\mathcal{V}(t.R)$. It takes linear time in the number of cliques to run the pivot clique selection heuristic. (Remark: If the efficiency of the heuristic cannot be ensured, then picking an arbitrary edge as pivot clique takes only constant time.) A pivot clique has at least two vertices, so we can cut at least two branches of each node according to Theorem 4.1. The size of the original search tree for a Bayesian network with n variables is $n!$. But pivot clique pruning can be applied in a recursive manner, because each pruning is guaranteed to produce a reduced search tree that has at least one optimal solution. As a result, the size of the reduced search tree is smaller than or equal to $(n-2)!$. To conclude, pivot clique pruning reduces the size of the search space by a factor of $\mathcal{O}(n^2)$, while the overhead cost for the pruning can be extremely low.

Algorithm 5 Depth-first search with pivot clique pruning.

```
1: Insert lines 1–10 of Algorithm 1
2: procedure ExpandNode(t, &best, &map)
3:     for all v ∈ V(t.R)\ SelectPivotClique(t.C(R)) do   ▷ Prune due to Theorem 4.1
4:         Let m = Copy(t)
5:         EliminateVertex(m, v)
6:         EliminateSimplicial(m)
7:         Insert lines 16–29 of Algorithm 1
8:     end for
9: end procedure
```

For the triangulation algorithm with treewidth as an objective, Bodlaender et al. [Bodlaender et al., 2012] proposed a similar pruning rule. There are some significant differences between their algorithm and the one proposed here. First, the algorithm in [Bodlaender et al., 2012] selects a maximum clique as a pivot

clique before the searching starts, and then uses the fixed pivot clique to prune unnecessary branches. In contrast, our proposed algorithm selects a pivot clique at each node expansion. Secondly, the method in [Bodlaender et al., 2012] prunes unnecessary orders on the basis of treewidth optimality. However, our method prunes unnecessary orders on the basis of total table size optimality.

4.3 Experiments

We conducted computational experiments to examine the effect of pivot clique pruning on the depth-first search algorithm. Our analysis uses repository Bayesian Networks and random Bayesian networks generated by BNGenerator software [Ide, 2015].

4.3.1 Naive depth-first search with pivot clique pruning

We first evaluated the reduction of the search space by applying the pivot clique pruning. As described in Chapter 2, pruning (or branch and bound) is very important for the performance of the depth-first search algorithm; however, pruning is dependent on both the quality of bounds and the graph topology of the Bayesian network. Therefore, we turned off branch and bound pruning to eliminate this confounding factor from our analysis of the effect of pivot clique pruning. We compared the number of node expansions and the running time for each of the following two algorithms.

- NDFS: the naive depth-first search (NDFS) algorithm that is obtained by turning off all the pruning in the Ottosen and Vomlel depth-first search algorithm.

- NDFS-PCP: the NDFS algorithm with pivot clique pruning.

The NDFS algorithm explores the search space of all elimination orders. In contrast, the NDFS-PCP algorithm removes a large number of redundant orders from the search space, and so it is expected to run faster than the NDFS algorithm. For the comparison of the two algorithms, we generated a set of random Bayesian networks with 12 variables and various treewidths using the BNGenerator software.

Table 4.1 shows the number of node expansions and the running time for each algorithm. The main observation is that the reduction of the number of node expansions by pivot clique pruning is more effective on graphs with larger treewidths. We observed that the graphs with larger treewidths tend to have higher density and thus are expected to have more large cliques. Therefore, pivot clique pruning can remove more branches from each node expansion for dense graphs. For the running times, it is clear that the NDFS-PCP algorithm has a larger speed advantage on graphs with larger treewidths. Because pivot clique pruning is more effective on graphs with larger treewidths, the NDFS-PCP algorithm prunes more nodes from the search space and runs faster than the NDFS algorithm does.

Table 4.1: A comparison of the running times (s) and the numbers of node expansions for the NDFS and the NDFS-PCP methods.

Bayesian Networks				Time			Nodes		
V	E	Density	tw	DFS,PCP	DFS	DFS/DFS,PCP	DFS,PCP	DFS	DFS/DFS,PCP
12	17	0.25	1	1.4492	2256.5781	1557.12	299242	1302061345	4351.20
12	19	0.28	2	0.9456	2394.3665	2532.11	202545	1302061345	6428.50
12	24	0.36	3	0.5147	2522.4697	4900.85	75596	1302061345	17223.94
12	34	0.51	4	0.0638	2531.9401	39685.58	3839	1302061345	339166.80
12	40	0.6	5	0.0233	2732.4136	117270.97	987	1302061345	1319211.09
12	48	0.72	6	0.0078	2441.0739	312958.19	110	1302061345	11836921.32
12	49	0.74	7	0.0053	2467.1237	465495.04	65	1302061345	20031713.00
12	57	0.86	8	0.0024	2493.7305	1039054.38	16	1302061345	81378834.06

4.3.2 Depth-first search with pivot clique pruning

In order to examine the effectiveness of the pivot clique pruning for the state-of-the-art algorithm, we compared the following two algorithms.

- DFS: the depth-first search algorithm with the proposed dynamic clique maintenance.
- EDFS : DFS with pivot clique pruning.

Since our proposed dynamic clique maintenance has been shown to be faster than other methods, both the DFS and EDFS algorithms use it internally for updating cliques. We empirically compared the two algorithms with respect to running time, number of expanded nodes and required space.

Table 4.2: A comparison of the running times (s), the numbers of expanded nodes and the sizes of coalescing maps for DFS and EDFS algorithms. The columns labeled with $mean(|sp|)$ and $sd(|sp|)$ give the average number of states of variables in each Bayesian network and the standard deviation, respectively. Finally, tw denotes the treewidth, and w-tw denotes the weighted treewidth.

Bayesian Networks									DFS			EDFS			Time
name	V	E	density	$mean(\|sp\|)$	$sd(\|sp\|)$	tw	w-tw	tts	Time (s)	Nodes	Map	Time (s)	Nodes	Map	$DFS/EDFS$
child	20	30	0.157	3	1.17	3	144	642	0.003	6	5	0.002	4	3	1.5
Insurance	27	70	0.199	3.3	0.99	6	4800	23880	0.696	4818	2291	0.199	3096	1919	3.49
water	32	123	0.247	3.62	0.49	9	589824	3028305	3.859	14438	6816	1.078	8049	5187	3.57
mildew	35	80	0.134	17.6	27.01	4	805200	3400464	4.209	69310	22351	0.668	15349	5222	6.3
alarm	37	65	0.097	2.84	0.73	4	108	996	0.003	35	27	0.002	27	25	1.5
Barley	48	126	0.111	8.77	9.05	7	6350400	17140796	*	*	*	2528.636	18824900	5566501	*
Hailfinder	56	99	0.064	3.98	1.72	4	3267	9406	3.75	44270	19650	1.655	31289	12537	2.26
WIN95PTS	76	225	0.078	2	0	8	512	2684	24.812	74227	34993	5.988	32084	14669	4.14
pathfinder	109	208	0.035	4.11	5.91	7	32256	182641	0.015	30	19	0.004	22	16	3.75

We ran the two triangulation algorithms on nine benchmark Bayesian networks. The results are presented in Table 4.2. The Time column lists the running time of the algorithms on these networks. The Nodes column gives the number of nodes expanded in the algorithms. The Map column gives the size of the coalescing map, which estimates the memory-consumption of the algorithms. A "*" indicates the algorithm did not finish within the time limit (one hour). Finally, the last column lists the ratio of the running time of DFS to that of EDFS. We observed that EDFS has from 1.5 to 6.3 times the speed of DFS. The two triangulation algorithms employ the same dynamic clique maintenance, but EDFS provides better performances than DFS. EDFS expanded fewer search nodes than DFS, because the pivot clique pruning can remove a lot of equivalent nodes from the search tree. The results show that reducing the number of expanded nodes effectively contributes to the reduction of the running time. For example, on the Mildew Bayesian network EDFS explored only 15,349 nodes, but DFS explored 69,310 nodes. As a result, EDFS improved the running time from 4.209 seconds to 0.668 seconds. For the Barley network, EDFS is the only algorithm that can find an optimal triangulation within the time limit. In addition, pivot clique pruning also leads to a considerable reduction of memory use (see the reduction of the size of coalescing map), which is also due to the reduction of the search tree.

Several graph parameters might influence the speed advantage of EDFS over DFS for triangulation of a Bayesian network, including the number of variables, the number of edges, the density of the moral graph, the average number of states of variables, the standard deviation of the number of states of variables, treewidth and weighted treewidth. To investigate the factors affecting the triangulation time, we analyzed the correlation between those factors and the speed-up of EDFS over DFS. Figure 4.2 depicts the results. The most important factors for determining the speed-up are the weighted treewidth, the *tts*, the average number of states of variables and the standard deviation of the number of states of variables. The

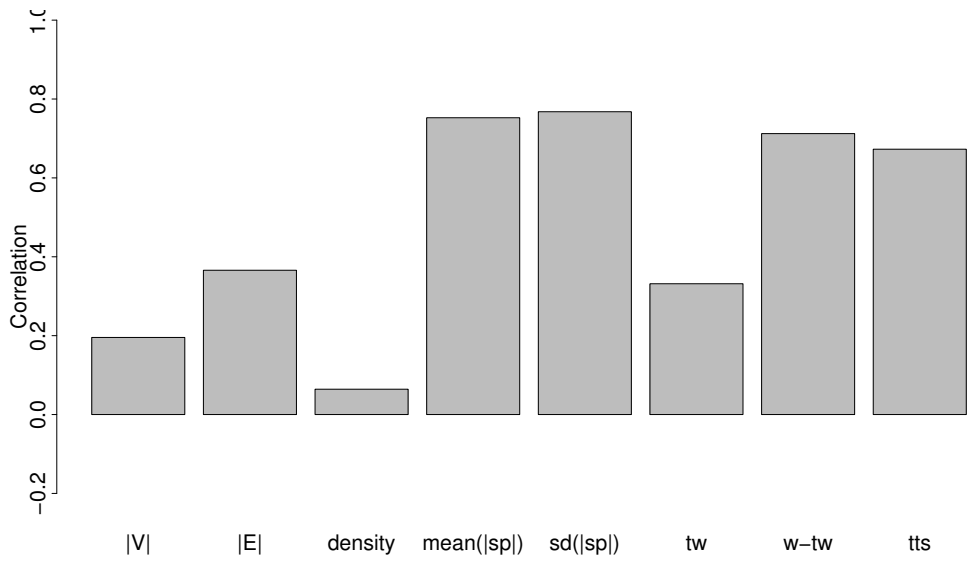


Figure 4.2: The correlation between the speed advantage of EDFS over DFS and several factors that might affect it.

correlation values between these factors and the speed-up are higher than 0.67. Additionally, all the correlation values are positive, indicating that there might be a higher speed-up when the Bayesian network has a more complex structure. This fact highlights the contribution of pivot clique pruning.

Our comparison between DFS and EDFS so far is based on the results for sparse graphs because the repository provides only a few sparse Bayesian networks. However, it is not clear how much improvement in running time can be obtained by EDFS for dense graphs. To answer this question, we generated a set of random graphs with various densities. In particular, we generated random graphs by adding some edges at random to the Insurance, Water and Alarm Bayesian networks. For each moral graph of the Bayesian network, we generated three random graphs with densities of 0.3, 0.4 and 0.5 (in total 9 random graphs). Because each group of three graphs has the same number of variables and their variables have the same state spaces, experiments on them can better demonstrate the performance of pivot clique pruning for various densities. Table 4.3 lists the running times of DFS and

Table 4.3: A comparison of DFS and EDFS for graphs with various densities.

BN	V	E	density	Time(DFS)	Time(EDFS)	DFS/EDFS
insurance3	27	106	0.3	86.548	28.388	3.04
insurance4	27	141	0.4	11.932	3.788	3.14
insurance5	27	176	0.5	3.814	1.123	3.39
water3	32	149	0.3	230.127	88.589	2.59
water4	32	199	0.4	95.226	32.054	2.97
water5	32	248	0.5	18.217	4.516	4.03
alarm3	37	200	0.3	8967.604	3277.388	2.73
alarm4	37	267	0.4	562.412	181.614	3.09
alarm5	37	333	0.5	66.777	16.416	4.06

EDFS for the random graphs. The last column of the table lists the ratio of the running time of DFS to that of EDFS. We also calculated the correlation between the time ratio and the density of graphs; this is 0.75. The result indicates that there is a higher speed-up when the Bayesian network has a denser graph. The reason is that our pivot clique pruning works well on dense graphs, because dense graphs tend to have more large cliques and then the more branches are pruned by the larger pivot cliques. As we explained in Section 3.3, for dense graphs, our proposed dynamic clique method does not improve the optimal triangulation algorithms much; however, pivot clique pruning works better on dense graphs.

4.3.3 Results for Bayesian networks with 100 variables

We compared EDFS with DFS (OandV) on a set of random Bayesian networks with 100 variables. The purpose of this experiment is to investigate the performance of our proposed algorithm on large scale networks. We generated ten random Bayesian networks for treewidth $k \in \{2, 5, 8\}$ respectively. These networks are

generated using the software BNGenerator. Each variable has a number of states randomly selected from 2 to 4. We ran EDFS and DFS on these Bayesian networks. The time limit for each instance is set to 1 h.

Table 4.4 reports the running time and the number of expanded nodes for each method. The "Time" column lists the running times of the algorithms on these networks. The "Nodes" column gives the number of nodes expanded by the algorithms. The mark "*" indicates the algorithm did not finish within the time limit (1 h). The results show that our proposed method is from 2.76 to 9.19 times the speed of DFS. For three networks with treewidth 5 and two networks with treewidth 8, the DFS cannot find an optimal triangulation within the time limit; in contrast, our proposed method does.

Table 4.4: A comparison of the EDFS and the DFS (OandV) algorithms on a set of Bayesian network with 100 variables

Bayesian Networks				Time			Nodes		
V	E	Density	tw	DFS (OandV)	EDFS	DFS/EDFS	DFS (OandV)	EDFS	DFS/EDFS
100	177	0.03	2	0.4229	0.1533	2.76	896	759	1.18
100	172	0.03	2	3.3153	1.1179	2.97	9146	8409	1.09
100	176	0.03	2	98.0187	25.2633	3.88	174406	124920	1.40
100	176	0.03	2	0.5754	0.0831	6.92	1090	452	2.41
100	178	0.03	2	0.0451	0.0072	6.26	140	69	2.03
100	180	0.03	2	0.0104	0.0019	5.47	13	10	1.30
100	166	0.03	2	1287.8544	300.2756	4.29	2758268	1683709	1.64
100	171	0.03	2	16.7556	3.8066	4.40	35908	26149	1.37
100	170	0.03	2	5.0397	1.4363	3.51	13156	9697	1.36
100	176	0.03	2	12.6262	3.2459	3.89	27965	20150	1.39
100	410	0.08	5	*	3265.6092	*	*	5339766	*
100	405	0.08	5	*	1967.2311	*	*	3640008	*
100	397	0.08	5	94.4848	15.9905	5.91	33271	31982	1.04
100	416	0.08	5	249.6761	27.1547	9.19	56721	44146	1.28
100	402	0.08	5	2937.612	330.2645	8.89	803643	622268	1.29
100	414	0.08	5	913.3883	106.7685	8.55	251592	211984	1.19
100	424	0.08	5	*	784.5601	*	*	1034457	*
100	628	0.12	8	2737.3103	314.8126	8.70	342487	338180	1.01
100	665	0.13	8	*	1941.9927	*	*	1213671	*
100	642	0.12	8	*	3160.7308	*	*	2564786	*

4.3.4 Triangulation with different objective functions

Table 4.5: A comparison of the different objective functions.

	EDFS,*tts*				EDFS,*tw*		EDFS,*w-tw*		EDFS,*fillin*		MinFill			
BN	*tts*	*tw*	*w-tw*	*fillin*	*tw*	*tts*	*w-tw*	*tts*	*fillin*	*tts*	*tts*	*tw*	*w-tw*	*fillin*
child	642	4	216	2	4	678	144	678	2	678	678	4	216	2
Insurance	23880	7	4800	26	7	29352	4800	29352	26	29352	29352	7	7200	26
water	3028305	10	589824	47	10	3657180	589824	3657180	46	3657180	3657180	11	1769472	47
Mildew	3400464	5	1249280	19	5	4434860	805200	4434860	19	4434860	4434860	5	1756800	19
alarm	996	5	108	5	5	1038	108	1038	5	1038	1038	5	144	5
Barley	17140796	8	7257600	46	8	17140796	6350400	17140796	45	17140796	17140796	8	7257600	46
HailFinder	9406	5	3267	17	5	9706	3267	9706	16	9706	9706	5	3267	16
Win95pts	2684	9	512	28	9	2684	512	2684	28	2684	2684	9	512	28
PathFinder	182641	7	32256	7	7	182641	32256	182641	7	182641	182641	7	32256	7

To perform efficient inference on a Bayesian network using the junction tree algorithm, we employed the total table size as the objective function to obtain the optimal triangulation of the Bayesian network. For general triangulation problems, the objective functions commonly have employed the treewidth, the weighted treewidth and the minimum number of fill-in edges. However, these objective functions are not guaranteed to optimize the total table size criterion. Therefore, this study assumes that directly optimizing the total table size improves the obtained triangulation of Bayesian networks. To ascertain this, we compared the performances of these objective functions with those of the total table size. Specifically, we performed EDFS, employing these objective functions on nine repository Bayesian networks and compared the total table sizes (tts) and the corresponding objective values (the treewidths (tw), the weighted treewidths (w-tw) and the minimum numbers of fill-in edges ($fillin$)) of the obtained triangulations with those of EDFS employing the total table size. In addition, we also applied the minimum fill-in heuristic (MinFill) on those networks to compare its performance for the obtained triangulations because it is a well-known heuristic that provides a good approximation to the exact solution (e.g., Gogate and Dechter [Gogate and Dechter, 2004]).

Table 4.5 shows the computational results. The main observation is that EDFS with tts as objective function (EDFS,tts) found triangulations with smaller total table sizes than the other methods did on six Bayesian networks. However, on Barley, Win95pts and PathFinder, our proposed algorithm EDFS,tts provided the same total table sizes as MinFill. Although MinFill just greedily selects the next vertex to eliminate, it works surprisingly well on the three networks. For Barley, Win95pts and PathFinder, EDFS,tts could use the exact optimal solution as an upper bound, since the MinFill provided the minimum total table sizes on the three networks. Taking advantage of using the tight initial upper bound, EDFS,tts was able to find an optimal total table size triangulation on PathFinder within 0.004 s

and on Win95pts within 5.988 s. On Barley, although EDFS,tts benefits from using the tight bound, surprisingly it took extremely long time (2528 s) to find an optimal total table size triangulation. This result suggests the importance of future work toward finding a good lower bound for the total table size. Interestingly, EDFS,tts also found the triangulations with the minimum treewidth for all the repository networks. This means that although the optimal total table size triangulation does not guarantee the minimum treewidth, it usually finds a triangulation with small treewidth.

For all the repository networks other than water, the triangulation found by MinFill also provided the minimum treewidth. Our results confirm the observation of Gogate and Dechter [Gogate and Dechter, 2004], that the minimum treewidth algorithm rarely finds better triangulations of the repository networks than MinFill does. In addition, MinFill also provided a good approximation to the minimum number of fill-in edges, which is obtained by EDFS,$fillin$. EDFS,w-tw provided a lower weighted treewidth than MinFill did.

Focusing on the total table size, the algorithms EDFS,tw, EDFS,w-tw and EDFS,$fillin$ provided the same total table sizes that MinFill did. However, EDFS with tts found better triangulations with smaller total table sizes for all the repository networks except for Barley, Win95pts and Pathfinder. Thus, the results demonstrate that employing the total table size improves the triangulations of Bayesian networks.

Chapter 5

Conclusions and future work

5.1 Conclusions

In this thesis an extended depth-first search (EDFS) algorithm for the optimal triangulation of Bayesian networks has been proposed. The new EDFS algorithm improves the state-of-the-art Ottosen and Vomlel (DFS) algorithm in two orthogonal directions: (1) reduction of the overhead cost and (2) reduction of the size of the search space. Theoretical analysis and experiments reveal that the EDFS algorithm is superior to the DFS algorithm. The EDFS algorithm lowers the time complexity of the Ottosen and Vomlel algorithm from $\mathcal{O}(\beta(n) \cdot n!)$ to $\mathcal{O}(\gamma(n) \cdot (n-2)!)$, where n is the number of vertices in the graph, and $\beta(n)$ and $\gamma(n)$ stand for the overheads for DFS and EDFS, respectively.

To reduce the overhead cost per node, a new algorithm for maintaining the cliques of a dynamic graph has been developed. The performance of the proposed algorithm was compared with the state-of-the-art Ottosen and Vomlel [Ottosen and Vomlel, 2012] and Li and Ueno [Li and Ueno, 2012] methods. The empirical results show that the new method is superior to the other methods for graphs with moderate size and density. By introducing the new dynamic clique maintenance, the overhead cost is reduced from $\beta(n)$ to $\gamma(n)$.

To reduce the number of nodes in the search tree, the idea of pivot clique pruning was introduced, and the pivot clique pruning theorem was proved in Section 4.2. In a theoretical analysis, we showed that the pruning reduced the size of the search tree from $n!$ to $\mathcal{O}((n-2)!)$. The reduction of the search tree achieved by introducing pivot clique pruning contributes effectively to the reduction of the running time of the optimal triangulation algorithm. If we do not apply any other pruning techniques, such as branch and bound, coalescing map pruning and simplicial vertex rule pruning, pivot clique pruning will at least cut the number of nodes by $(n! - (n-2)!)$. In this case, our pruning provides $n(n-1)$ times speed-up over the original algorithm. However, it is difficult to analyze the time complexity of the optimal triangulation algorithm combining all these smart pruning techniques. Nevertheless, experiments show that EDFS is 1.3 to 6.3 times the speed of DFS (with the proposed dynamic clique maintenance) for the repository datasets. The pivot clique pruning also permits a considerable reduction in space requirements, which is also due to the reduction of the search space.

5.2 Future work

Although our two proposed methods contributed to improvements in the running time and scalability of the optimal triangulation algorithms, the algorithms are still limited to relatively small and sparse Bayesian networks. Nevertheless, exact optimal triangulation algorithms are valuable because the optimal triangulation enables time-efficient inference using the junction tree algorithm. Optimal triangulation requires additional work time, but once the triangulation of a Bayesian network has been done off-line, propagation can be done many times on the same junction tree to process any evidence. In addition, total table size is important in estimating the running time for inference on Bayesian networks. In the study of the relationship between the junction tree inference time and the structure of the

Bayesian network, Ole J. Mengshoel used a heuristic triangulation that obtained an approximate total table size to estimate inference time [Mengshoel, 2010]. Our study on optimal triangulation might improve Mengshoel's results. Finally, the empirical results in Section 4.3 show that the running time of the optimal triangulation on large Bayesian networks is difficult to predict. It would be interesting to characterize the hardness of finding an optimal triangulation for a given Bayesian network.

Bibliography

Emgad H. Bachoore and Hans L. Bodlaender. A branch and bound algorithm for exact, upper, and lower bounds on treewidth. In *Proceedings of the Second International Conference on Algorithmic Aspects in Information and Management*, AAIM'06, pages 255–266, Berlin, Heidelberg, 2006. Springer-Verlag.

Emgad H. Bachoore and Hans L. Bodlaender. Weighted treewidth algorithmic techniques and results. In *International Symposium on Algorithms and Computation*, pages 893–903. Springer, 2007.

Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. *ACM Trans. Algorithms*, 9(1):12:1–12:23, 2012. ISSN 1549-6325.

Hans L. Bodlaender, Arie M.C.A. Koster, and Frank van den Eijkhof. Preprocessing rules for triangulation of probabilistic networks. *Computational Intelligence*, 21(3):286–305, 2005.

F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407:564 – 568, 2008. ISSN 0304-3975.

Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

P. Alex Dow and Richard E. Korf. Best-first search for treewidth. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1146–1151. AAAI Press, 2007. ISBN 978-1-57735-323-2.

Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 201–208, Arlington, Virginia, United States, 2004. AUAI Press. ISBN 0-9749039-0-6.

Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. North-Holland Publishing Company, 2 edition, 2004.

Jaime Shinsuke Ide. BNGenerator. `http://sites.poli.usp.br/pmr/ltd/`, 2015.

Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990. ISSN 0723-712X.

Finn Verner Jensen and Frank Jensen. Optimal junction trees. In *UAI*, pages 360–366, 1994.

U Kjaerulff. Triangulation of graphs – algorithms giving small total state space. *Technical Report R9009 Department of Mathematics and Computer Science Aalborg University Denmark*, 1990.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):pp. 157–224, 1988. ISSN 00359246.

Chao Li and Maomi Ueno. A depth-first search algorithm for optimal triangulation of Bayesian network. In *Proceedings of the Sixth European Workshop on Probabilistic Graphical Models*, pages 187–194, 2012.

Anders L Madsen and Finn V Jensen. Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1):203–245, 1999.

Ole J. Mengshoel. Understanding the scalability of Bayesian network inference using clique tree growth curves. *Artificial Intelligence*, 174:984 – 1006, 2010. ISSN 0004-3702.

D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul. The challenges of real-time AI. *Computer*, 28(1):58–66, 1995.

Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian Networks and Decision Graphs*. Information Science and Statistics. Springer-Verlag New York, 2 edition, 2007.

Thorsten Ottosen and Jiri Vomlel. All roads lead to rome: New search methods for the optimal triangulation problem. *International Journal of Approximate Reasoning*, 53(9):1350–1366, 2012. ISSN 0888-613X.

S. Parter. The use of linear graphs in gauss elimination. *SIAM Review*, 3(2): 119–130, Apr. 1961.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1988. ISBN 1-55860-479-0.

Fabio Tozeto Ramos and Fabio Gagliardi Cozman. Anytime anyspace probabilistic inference. *International Journal of Approximate Reasoning*, 38(1):53 – 80, 2005. ISSN 0888-613X.

Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82: 273 – 302, 1996. ISSN 0004-3702.

Marco Scutari. Bayesian Network Repository. `http://www.bnlearn.com/bnrepository/`, 2016.

Prakash P. Shenoy and Glenn Shafer. Axioms for probability and belief-function propagation. In *Uncertainty in Artificial Intelligence*, pages 169–198, 1990.

Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28 – 42, 2006. ISSN 0304-3975. Computing and Combinatorics10th Annual International Conference on Computing and Combinatorics (COCOON 2004).

Frank van den Eijkhof, Hans L. Bodlaender, and MC Arie Koster. Safe reduction rules for weighted treewidth. *Algorithmica*, 47(2):139–158, 2007.

Wilson Wen. Optimal decomposition of belief networks. In *Proceedings of the Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pages 245–256, New York, NY, 1990. Elsevier Science.

David R. Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23(3):337–352, 2007.

List of Publications

Journal Papers

1. Chao Li and Maomi Ueno. An extended depth-first search algorithm for optimal triangulation of Bayesian networks. International Journal of Approximate Reasoning, 80:294-312, 2017. ISSN 0888-613X, http://dx.doi.org/10.1016/j.ijar.2016.09.012.

International Conferences (Refereed)

1. Chao Li and Maomi Ueno. A Depth-First Search Algorithm for Optimal Triangulation of Bayesian Network. In Proceedings of the Sixth European Workshop on Probabilistic Graphical Models (PGM 2012), pages 187-194, 2012.

2. Chao Li and Maomi Ueno. A Fast Clique Maintenance algorithm for Optimal Triangulation of Bayesian Networks. In Proceedings of the Second International Workshop on Advanced Methodologies for Bayesian Networks (AMBN 2015), LNAI 9505, pages 152-167, 2015