An extended depth-first search algorithm for optimal triangulation of Bayesian networks

Article *in* International Journal of Approximate Reasoning · September 2016
DOI: 10.1016/j.ijar.2016.09.012

2 authors:

Chao Li
The University of Electro-Communications
3 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Maomi Ueno
The University of Electro-Communications
121 PUBLICATIONS **377** CITATIONS
SEE PROFILE

International Journal of Approximate Reasoning 80 (2017) 294–312

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

An extended depth-first search algorithm for optimal triangulation of Bayesian networks

Chao Li *, Maomi Ueno

The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan

ARTICLE INFO

Article history:
Received 16 February 2016
Received in revised form 18 July 2016
Accepted 28 September 2016
Available online 30 September 2016

Keywords:
Bayesian network
Probabilistic inference
Optimal triangulation
Dynamic clique maintenance

ABSTRACT

The junction tree algorithm is currently the most popular algorithm for exact inference on Bayesian networks. To improve the time complexity of the junction tree algorithm, we need to find a triangulation with the optimal total table size. For this purpose, 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 that algorithm leave much room for improvement. First, the dynamic clique maintenance allows to recompute some cliques. Second, in the worst case, the DFS algorithm explores the search space of all elimination orders, which has size $n!$, where n is the number of variables in the Bayesian network. To mitigate these problems, we propose an extended depth-first search (EDFS) algorithm. The new EDFS algorithm introduces the following two techniques as improvements to the DFS algorithm: (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.

1. Introduction

Bayesian networks are graphical models that encode probabilistic relations among several variables [1]. 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 variable is associated with a conditional probability table conditioning on its parents, which quantifies the relation between the variable and its parents. Bayesian networks provide a neat and compact representation of joint probability distributions.

Probabilistic inference is an extremely common task that is conducted on Bayesian networks. However, exact computation of posterior marginal distributions in a Bayesian network is known to be NP-hard [2] and even computing an approximation is computationally intractable in the general case [3]. 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

* Corresponding author.
E-mail address: chao.li.314@gmail.com (C. Li).

http://dx.doi.org/10.1016/j.ijar.2016.09.012
0888-613X/

have been made in the past two decades. The junction tree algorithm [4–6] 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 triangulated. If the graph is not triangulated, then we must add extra edges to it until it becomes so. This process is called triangulation and, in general, any Bayesian network allows several different triangulations. The triangulation will affect the structure of the junction tree and the performance of subsequent belief propagation on that tree. So, to perform efficient inference on a Bayesian network by using the junction tree algorithm, we aim to find a triangulation of the moral graph of the Bayesian network with the minimum total table size [7,8]. Unfortunately, finding a triangulation with the minimum total table size is NP-hard [9]. Due to the complexity, early research in this direction focused mainly on developing approximation algorithms, such as greedy heuristics [7,9]. The heuristic approaches are useful for triangulation of large-scale Bayesian network, 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 of a Bayesian network has been constructed, we can perform efficient probabilistic inference on the same junction tree to process any evidence [10,11]. 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 real-time systems [12] and embedded systems [13]) requires an optimal triangulation to minimize the inference time. Therefore, in this paper, we focus 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 [14], branch and bound [15], best-first search [16] and dynamic programming [17]. Instead of using the total table size as a measure, these methods have employed treewidth. The treewidth of a triangulated graph is the maximum clique size minus one. A junction tree is constructed by connecting the (maximal) cliques of a triangulated graph. The time that a belief propagation takes to process one 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. 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. However, the weighted treewidth of a triangulated 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 [18,19]. Nevertheless, when all cliques are not almost 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. Therefore, of all these optimality criteria, the total table size yields the most exact bound for the time requirement of probabilistic inference [4,9,8]. Thus, for inference on a Bayesian network, a triangulation is optimal when it has the minimum total table size.

A triangulation can be found by an algorithm called elimination. In this algorithm, a triangulated graph is obtained by eliminating all vertices from a graph according to a linear ordering of the vertices of the graph (called elimination order). It is well known that the optimal triangulation problem can be formulated as a problem to find an elimination order such that the triangulated graph obtained according to the ordering gives 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 [20]. 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.) We focus mainly on improvements to depth-first search algorithm for optimal triangulation.

In the depth-first search algorithm, in order to employ the branch and bound method 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 associated with each node. A simple method for computing this is to run the Bron–Kerbosch (BK) algorithm [21] for the graph belongs to each node; however, the complexity of the BK algorithm is exponential in the number of vertices of the graph [22]. To resolve this problem, Ottosen and Vomlel proposed a dynamic clique maintenance algorithm [20] 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 [23,19] and coalescing map pruning [16,11]. 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 [24] 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 time complexity of the depth-first search algorithm is $\mathcal{O}(n!)$, where n is the number of variables in the Bayesian network, because it explores a 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 great number of equivalent elimination orders.

In this paper, we propose an extended depth-first search algorithm for the optimal triangulation of Bayesian networks. The 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, we propose a new dynamic clique maintenance algorithm. When new edges are inserted in a graph during triangulation, 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. We run the BK algorithm on the subgraph that contains only the vertices connected by new edges and all neighboring vertices of new edges. We, therefore, explore 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, we introduce a novel pruning rule, called pivot clique pruning. 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, we show 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.

For the triangulation algorithm with treewidth as an objective, Bodlaender et al. [17] proposed a similar pruning rule. There are some significant differences in algorithms between that in [17] and the one proposed here. First, the algorithm in [17] 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. Second, the method in [17] prunes unnecessary orders on the basis of treewidth optimality. However, our method prunes unnecessary orders on the basis of total table size optimality.

The rest of this paper is organized as follows. Section 2 introduces the junction tree algorithm and clarifies the significance of the triangulation. Section 3 introduces the triangulation problem and describes the formulation of the search space of the depth-first search optimal triangulation algorithm. Section 4 reviews the Ottosen and Vomlel optimal triangulation algorithm [20]. In Section 5, we propose two techniques for improving the Ottosen and Vomlel algorithm: a new dynamic clique maintenance algorithm and pivot clique pruning. Section 6 discusses some experiments to evaluate the proposed method against the state-of-the-art method. Section 7 concludes the paper.

2. Background

A Bayesian network is a directed acyclic graph (DAG) in which the set of vertices corresponds to a set of random variables $X = \{x_1, x_2, \ldots, x_n\}$, and the arcs represent direct dependency relations between the variables. For example, we show the classical Asia Bayesian network [5] in Fig. 1. 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), $P(x_i|PA_i)$, where PA_i denotes the parents of x_i in the DAG. The product of CPTs in a Bayesian network gives the joint probability distribution of variables in the Bayesian network, with

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

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

When an inference task is performed on a Bayesian network, we typically compute 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 works in two processes: compilation and propagation. The compilation part of the method consists of the following steps:

1. moralize the Bayesian network graph, see Fig. 2;
2. triangulate the moralized graph (i.e., add extra edges such that every cycle of length greater than three has a chord), see Fig. 3(a);
3. identify all cliques of the triangulated graph;
4. construct a junction tree over these cliques, see Fig. 3(b).

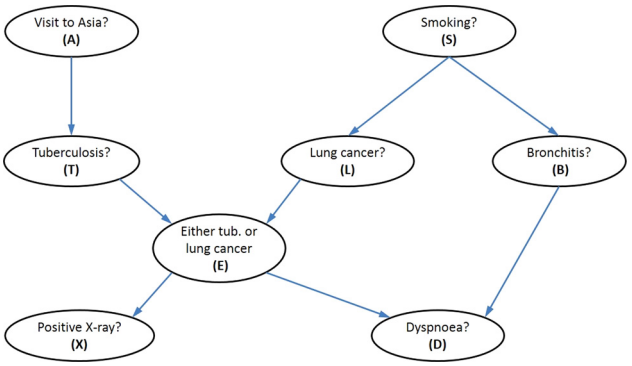


Fig. 1. The Asia Bayesian network.

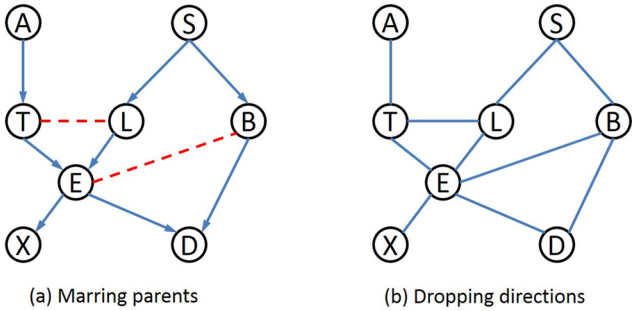


Fig. 2. Moralizing the Bayesian network graph: (a) connect the vertices with common children, and (b) drop directions of directed edges.

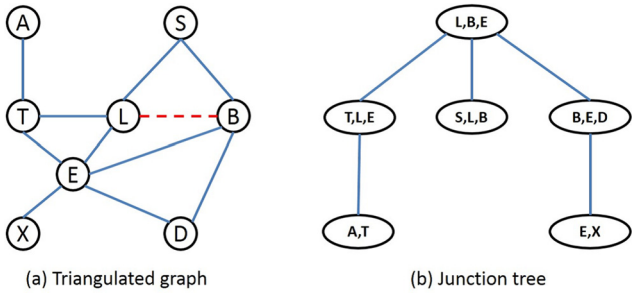


Fig. 3. (a) Add edges to make the moral graph triangulated, and (b) construct a junction tree by connecting the cliques of the triangulated graph.

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 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 paper focuses on the optimal triangulation algorithm. For step 4, Jensen [25] has proposed an algorithm for optimal junction tree construction.

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 Fig. 4(a));
2. forming a potential ϕ_i for each clique C_i, using the CPTs of the Bayesian network and attaching a potential ϕ_{ij} to all separators (initialize all values to one); and
3. letting the nodes communicate via the separators. For example, see Fig. 4(b), a message sent from clique C_i to C_j with separator S_{ij} has the form that ϕ_i is marginalized down to S_{ij}, resulting in ϕ'_{ij}; the message ϕ'_{ij}/ϕ_{ij} is received by ϕ_j, and ϕ_{ij} on the separator is replaced by 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. This continues until the root clique has

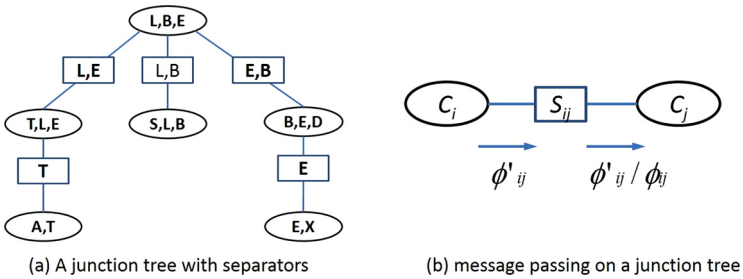


Fig. 4. (a) Junction tree and (b) an illustration of message passing.

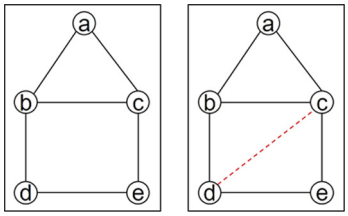


Fig. 5. Left: Initial graph $G = (V, E)$. Right: Updated graph G' obtained by adding one edge (c, d) to G.

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 in the case of only binary variables, performing probabilistic inference on the tree needs to calculate the number $\sum_{i=1}^{m} 2^{m_i}$ of parameters, where m_i denotes the number of variables in the i-th clique. The number $\sum_{i=1}^{m} 2^{m_i}$ is known as the total table size (or total clique tree size [26] or total state space size [7]), which 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 triangulated graph. Therefore, it is necessary to find an optimal total table size triangulation for efficient inference.

3. 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.

3.1. 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 $W \subseteq V$, $G[W] = (W, \{(v, w) \in E | v, w \in W\})$ 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 neighbors of v in graph $G = (V, E)$ are 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 $(\cup_{u \in U} \mathcal{N}(u, G)) \cup U$.

A graph G is complete if all pairs of vertices $(u, v)(u \neq v)$ are adjacent in G. A set of vertices $W \subseteq V$ 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 of new edges F 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. 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 $\mathcal{NC}(G, G') = \{\{b, c, d\}, \{c, d, e\}\}$.

The table size of a clique C 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 $tts(G) = \sum_{C \in \mathcal{C}(G)} ts(C)$.

An undirected graph G is triangulated if every cycle of length greater than three has a chord, that is an edge connecting two nonconsecutive vertices in the cycle. A triangulation of G is defined as adding a set of edges T such that $T \cap E = \emptyset$ and graph $H = (V, E \cup T)$ is triangulated. For example, in Fig. 5, the graph on the left is not triangulated because a chord-less

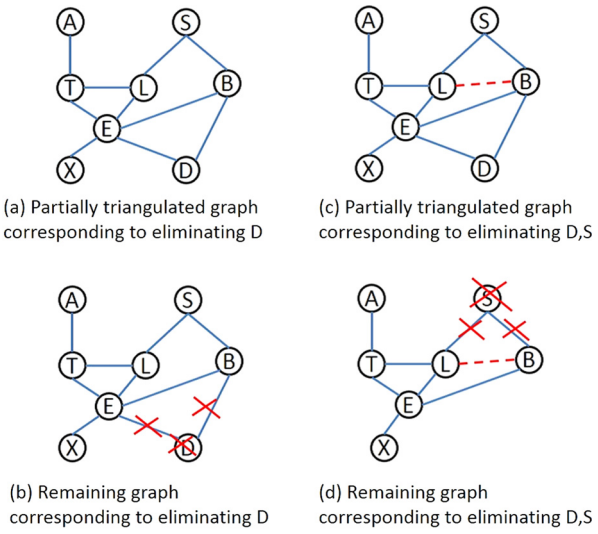


Fig. 6. An example of eliminating vertices from the moral graph of the Asia network.

cycle $\{b, c, e, d\}$ exists. The graph on the right is triangulated, and the edge (c, d) is 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, 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 $\pi : \{1, 2, \ldots, |V|\} \to V$ describing an order for eliminating all vertices from G, where $\pi(i)$ denotes the i-th 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 triangulated graph [27]. Now, it suffices to define the partially triangulated graph H_i^π 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 triangulated 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 Fig. 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 edge, 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 Fig. 6(a)) and a remaining graph G_2^π (see Fig. 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 Fig. 6(c) and the remaining graph G_3^π is shown in Fig. 6(d). If we continue to eliminate vertices until no vertex is left, the final partially triangulated graph H^π is a triangulated 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 paper, we try to find the order π for eliminating graph G that induces a triangulated graph H^π with the minimum total table size.

3.2. 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 [20]. 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. Fig. 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 inducing a triangulated graph with the minimum total table size.

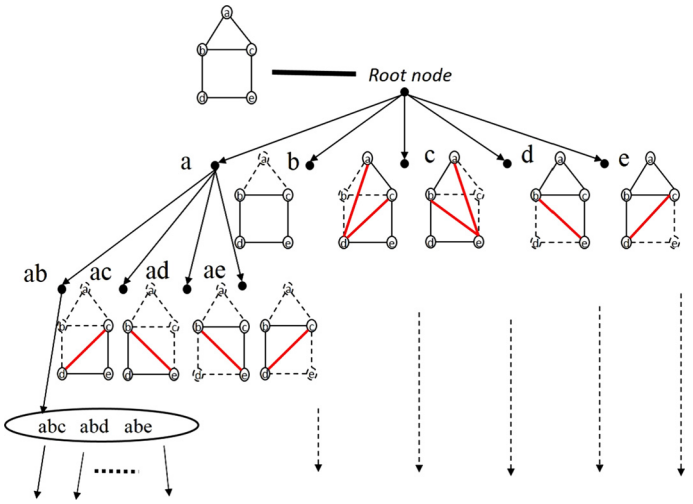


Fig. 7. The search tree of the optimal triangulation algorithm for a network graph with five vertices.

4. The optimal triangulation algorithm

This section reviews the depth-first search optimal triangulation algorithm presented by Ottosen and Vomlel [20].

4.1. Depth-first search algorithm

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) [7,9,11], which greedily selects the next vertex to eliminate if the elimination adds the minimum number of fill-in edges. 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 on the tts of the node because the tts of a graph cannot be decreased by adding edges [20]. If we find a node of 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 of which the tts is smaller than UB, we update UB by replacing UB with the leaf node (including the triangulated 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 efficiently, Ottosen and Vomlel associate the following with each node t [20].

- $t.\tau$: 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 of cliques for H, $\mathcal{C}(H)$.
- $t.tts$: The total table size of graph H, which is a lower bound on the tts of node t.
- $t.R$: The remaining graph, $R = H[V \setminus \mathcal{V}(\tau)]$, 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 algorithm (BK algorithm) [21]. Unfortunately, the BK algorithm has a heavy computational cost. For a graph with n vertices, the worst-case running time of the BK algorithm is $\mathcal{O}(3^{n/3})$ [22]. Indeed, eliminating one vertex changes only a small part of a partially triangulated graph. Performing the BK algorithm on the whole graph thus involves many redundant computations. To tackle this problem, Ottosen and Vomlel [20] 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 4.2. However, the dynamic clique maintenance algorithm proposed by Ottosen and Vomlel allows to compute some duplicate cliques. To resolve this problem, we propose a new dynamic clique maintenance algorithm in Section 5.1.

We have explained the search tree of depth-first search and how to compute a lower bound on the tts for each node. The depth-first search algorithm presented by Ottosen and Vomlel can be implemented in $O(|V|)$ space and $O(|V|!)$ time. A pseudo code for the Ottosen and Vomlel algorithm is outlined in Algorithm 1. The procedure EliminateVertex(m, v)

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
```

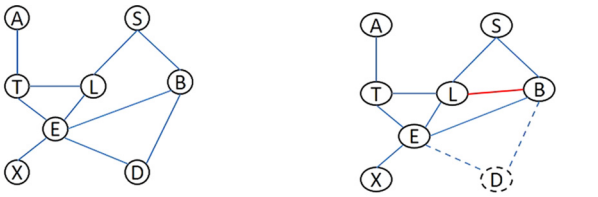
eliminates vertex v from the remaining graph of node m and simultaneously updates the set of cliques and the total table size. To prune unnecessary search nodes further, Ottosen and Vomlel also introduced the following pruning rules: (1) a graph reduction technique called the simplicial vertex rule [23,19], and (2) coalescing of nodes [16,11]. The procedure EliminateSimplicial(m, v) sequentially removes all simplicial vertices from the remaining graph of node m. The coalescing map uses $\mathcal{O}(2^n)$ memory space to prune unnecessary search nodes (see [16,11,20] for details). Although it is well known that a depth-first search algorithm runs in $O(|V|!)$ time, the algorithm combined with the pruning techniques described above merely hits the upper bound. However, in the worst case, the algorithm might explore the search space of all elimination orders which has a size of $n!$, where n is the number of variables in the Bayesian network. It is known that some different elimination orders induce identical triangulations. Consequently, the Ottosen and Vomlel algorithm might explore a great number of equivalent elimination orders. In Section 5.2, we propose a pruning rule called pivot clique pruning, which removes a large number of unnecessary elimination orders from the search space.

4.2. 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 [20]. This dynamic clique maintenance is presented in Algorithm 2,



(a) Partially triangulated graph corresponding to eliminating D

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

Fig. 8. A sequence of graphs corresponding to eliminating vertices D and then S. (L, B) is the fill-in edge.

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 1 *([20]). 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 $U = \mathcal{V}(F)$, and let $\mathcal{C}(G')$ be initialized with $\mathcal{C}(G)$. If the cliques in G those are intersect with U are removed from $\mathcal{C}(G')$ and the cliques in $G'[\mathcal{FA}(U, G')]$ those are 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 1. Consider the Fig. 8(a). $\mathcal{C}(G)$ 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 Fig. 8(b)). 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 [24] 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

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

```
1:  procedure CliqueUpdate1(G, C(G), F, W)
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:              if C ∩ W = ∅ then
8:                  Set C(G') = C(G') \ {C}
9:              end if
10:         end if
11:     end for
12:     let C^new = BKalgorithm(G'[FA(U, G') \ W])
13:     for each clique C ∈ C^new do                          ▷ Add new cliques
14:         if C ∩ U ≠ ∅ then
15:             Set C(G') = C(G') ∪ {C}
16:         end if
17:     end for
18: end procedure
```

necessary to ascertain whether clique C and W are disjoint. This check is usually not a problem because the complexity of comparison of 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.

5. New algorithm

5.1. Proposed dynamic clique maintenance algorithm

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
```

In the depth-first search optimal triangulation algorithm, it is necessary to compute the lower bound on the *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 4.2, 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 in 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 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 2. Consider again the graph G and updated graph G' in Fig. 8. $\mathcal{C}(G)$ 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 $\mathcal{C}^{new}$ 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 $\mathcal{C}(G')$. On the other hand, 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 $\mathcal{V}(F)$ or not. However, we can remove this conditional check from our algorithm.

The dynamic clique maintenance algorithms work in two main steps: scanning all existing cliques and running the BK algorithm. If dynamic clique maintenance algorithms are used for computing graphs with 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 Bayesian networks with less than 100 vertices found that there are not many cliques with 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 Section 6.

5.2. Pivot clique pruning

In the worst case, the depth-first search algorithm described in Section 4.1 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 lead to identical triangulations. Consequently, the depth-first search algorithm might unnecessarily explore a great number of elimination orders. In order to prune these redundant elimination orders, we propose a novel pruning rule called pivot clique pruning, which can remove a large number of unnecessary elimination orders from the search space.

We first show an example of elimination orders leading to duplicate results. Consider the process of eliminating vertices from the left graph G in Fig. 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 the triangulated graph H^{π}, which is shown as the right graph in Fig. 5. The order π' also leads the identical triangulated graph, $H^{\pi'} = H^{\pi}$. 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, 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, $\mathcal{N}(e, G_2^{\pi}) = \mathcal{N}(e, G_3^{\pi})$. Next, consider eliminating vertex $\pi'(2) = e$ from graph $G_2^{\pi'}$. Then 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. Because $\mathcal{N}(b, G_2^{\pi}) = \mathcal{N}(b, G_3^{\pi'})$, 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 equal: $G_4^{\pi} = G_4^{\pi'}$ and $G_5^{\pi} = G_5^{\pi'}$. Thus, the fill-in edges obtained from the orders π and π' are identical.

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 explicitly identify the equivalent orders. The following theorems offer a straightforward approach to prune redundant orders with extremely low computational cost.

Lemma 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}. If 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 $l = i, \ldots, k-2$, then by moving v_k directly before v_i to obtain the new order $\pi' = (v_1, \ldots, v_{i-1}, v_k, v_i, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$, we find that the orders π and π' engender identical triangulated graphs.*

Proof. First, we prove for $l = i, \ldots, k-1$, 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 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}. If G_i^{π} is not complete, there exists a vertex v_j $(i < j)$ that is not adjacent to v_i. Then, by moving v_j directly before v_i, we obtain an order $\pi' = (v_1, \ldots, v_{i-1}, v_j, v_i, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n)$ with a tts that is smaller than or equal to the tts of π.*

Proof. The sequence of vertices $(v_i, \ldots, v_n)$ 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 $l = i, \ldots, k-2$, or
2. v_l is adjacent to v_{l+1}, for $l = i, \ldots, n-1$.

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

Now, we prove the theorem for each case.

(a) In G_{m-1}, 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 $\omega = (v_1, \ldots, v_{m-1}, v_k, v_m, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$. 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 [20].

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 $l = i, \ldots, m-2$, from Lemma 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_{m+1}^{π} is complete. By moving v_k directly before v_{m+1}, we obtain order $\omega = (v_1, \ldots, v_m, v_k, v_{m+1}, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$. 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, $H^{\omega} = H^{\pi}$.

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 $l = i, \ldots, m-1$. From Lemma 1, $H^{\pi'} = H^{\omega}$. Therefore, the *tts* of $H^{\pi'}$ is equal to the *tts* of H^{π}. □

Now, using Lemma 2, the following pivot clique pruning theorem can be derived.

Theorem 3 *(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 Fig. 9. 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.

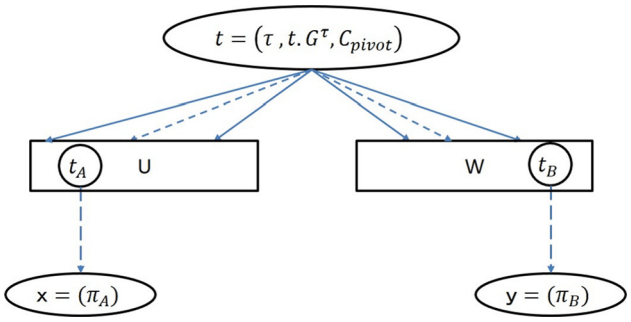


Fig. 9. The part of search tree beginning at node t.

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 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 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. Searching the largest clique of the remaining graph $t.R$ seems to be expensive, however, it can be easily computed by searching 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 we cannot ensure the efficiency of the heuristic, 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 3. The size of the original search tree for a Bayesian network with n variables is $n!$. Notice that 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 3
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
```

6. 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 HUJI repository.[1] We also generated a set of random graphs for doing controlled experiments. We compared our algorithm with state-of-the-art algorithms on the benchmark networks and the random graphs. All the algorithms described in this paper are implemented in the Java language.[2] The experiments were

[1] http://www.cs.huji.ac.il/site/labs/compbio/Repository/.

[2] A software package with source code named OptimalTriangulation implementing the proposed algorithm can be downloaded at http://www.ai.is.uec.ac.jp/optimaltriangulation/.

Table 1
A comparison of the running times (seconds) for the Ottosen and Vomlel method (OandV), the Li and Ueno method (LandU2012) and the proposed method.

Bayesian networks				Time (OandV)			Time (LandU2012)			Time (proposed)		
BN	V	E	Density	DCM	DFS	Total	DCM	DFS	Total	DCM	DFS	Total
Insurance	27	70	0.199	1.083	0.21	1.293	0.537	0.173	0.71	0.352	0.187	0.539
water	32	123	0.247	6.623	0.897	7.52	3.883	0.912	4.795	2.621	0.978	3.599
Mildew	35	80	0.134	8.647	1.576	10.223	4.511	1.516	6.027	2.541	1.697	4.238
alarm	37	65	0.097	0.007	0.005	0.012	0.002	0.002	0.004	0.002	0.002	0.004
HailFinder	56	99	0.064	5.319	1.37	6.689	3.325	1.391	4.716	1.981	1.35	3.331
Win95pts	76	225	0.078	36.117	7.846	43.963	23.637	7.683	31.32	14.058	7.672	21.73
PathFinder	109	208	0.035	0.017	0.012	0.029	0.01	0.003	0.013	0.008	0.003	0.011

performed on a Windows 8.1 PC with 2.6 GHz Intel Xeon Processor E5-2640 and 12GB RAM, running version 8 of the Java Virtual Machine.

6.1. Dynamic clique maintenance

We first compare the computational time of our proposed dynamic clique maintenance, described in Section 5.1, with those of state-of-the-art algorithms. For this purpose, we implemented the following algorithms.

- DFS (OandV): the depth-first search optimal triangulation algorithm proposed by Ottosen and Vomlel, which uses the Ottosen and Vomlel approach [20] for dynamic clique maintenance.
- DFS (LandU2012): the DFS algorithm with the Li and Ueno dynamic clique maintenance [24].
- DFS (proposed): the DFS algorithm with the proposed dynamic clique maintenance.

All the depth-first search optimal triangulation algorithms have a depth-first search part and a dynamic clique maintenance part. Therefore, we empirically compared the three algorithms with respect to the total running times, the depth-first search (DFS) time and the dynamic clique maintenance (DCM) time. We ran each algorithm to triangulate the eight graphs in the Bayesian network repository. Table 1 shows the results for the three algorithms. For the Barley Bayesian network, no triangulation algorithm completes the computation in one hour. Therefore, we reported the results for only seven Bayesian networks. For the total running times, it is clear that our proposed dynamic clique maintenance remarkably improves the running time of the optimal triangulation algorithm. We also observed that, for the Alarm and PathFinder Bayesian networks, each triangulation algorithm can find an optimal triangulation in less than 0.1 seconds. For DFS times, we observed that, for the seven benchmark Bayesian networks, the DFS times of the three algorithms are almost same. As a result, the differences among the computational times of the algorithms are only caused by DCM times. From the DCM time results, we see that the proposed method is faster than the LandU2012 method, and the LandU2012 method is faster than the OandV method. The explanation is that the BK algorithm has a heavy computational cost and the proposed method reduces the search space of the BK algorithm.

The comparison of dynamic clique maintenance algorithms on the random graphs is 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 1,000 triangulations of a graph with 25 vertices performed 25,000 dynamic clique maintenance steps. Therefore, we normalized this time by dividing 25. Fig. 10 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 is that there are too many cliques in a dense graph and so the scanning for cliques part 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 $\mathcal{O}(n \cdot 2^d)$ [28], 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 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.

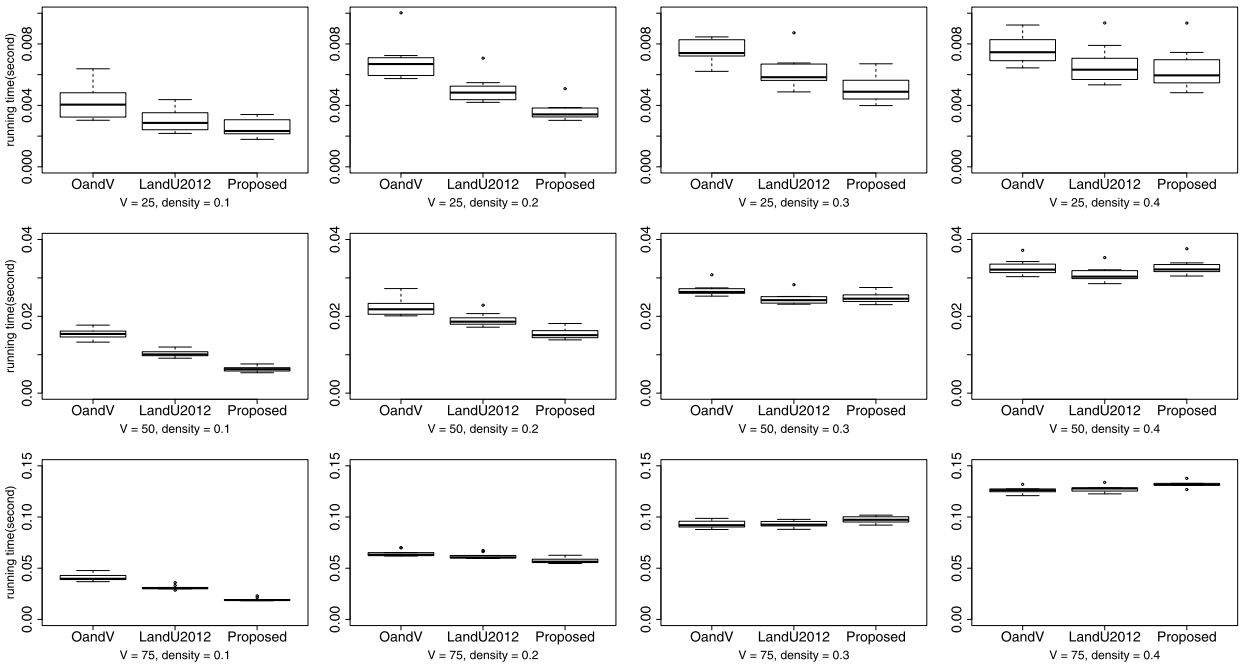


Fig. 10. A comparison of the running times for different dynamic clique maintenance algorithms.

Table 2
A comparison of the running times (seconds), 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 of that, 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

6.2. Depth-first search with pivot clique pruning

To examine the effectiveness of the pivot clique pruning method described in Section 5.2, 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 the running times, the number of expanded nodes and the space requirements. In particular, we compared space requirements by considering the size of the coalescing map, because it is the most space-consuming data structure that is used.

We performed the two triangulation algorithms on nine benchmark Bayesian networks. The results are presented in Table 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

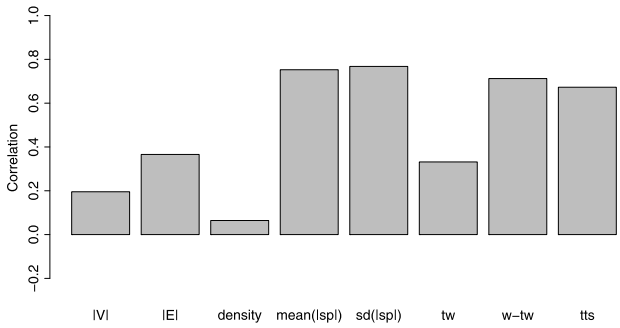


Fig. 11. The correlation between the speed-up of EDFS over DFS and several factors that might affect the speed-up.

Table 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

equivalent nodes from the search tree. We can see 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 affect the speed-up of EDFS over DFS for triangulation of a Bayesian network, including the number of variables, the number of edges, the density of moral graph, the average number of states of variables, the standard deviation of the number of states of variables, treewidth, and weighted treewidth. We analyzed the correlation between those factors and the speed-up of EDFS over DFS. Fig. 11 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, because all the correlation values between these factors and the speed-up are higher than 0.67. Additionally, all the correlation values are positive indicates 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 uniformly 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 3 lists the running times of DFS and 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, which 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 showed in Section 6.1, for dense graphs, our proposed dynamic clique method does not improve the optimal triangulation algorithms much; however, the pivot clique pruning works better on dense graphs.

6.3. Triangulation with different objective functions

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

Table 4
A comparison of the different objective functions.

BN	EDFS, *tts*				EDFS, *tw*		EDFS, *w-tw*		EDFS, *fillin*		MinFill			
	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

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 network. 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 performed 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 [14]).

Table 4 indicates 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 that MinFill did. 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 seconds and on Win95pts within 5.988 seconds. On Barley, although EDFS, tts benefits from using the tight bound, surprisingly it took extremely long time (2528 seconds) to find an optimal total table size triangulation. This result suggests the importance of future work toward finding a good lower bound on 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 except for water, the triangulation found by MinFill also provided the minimum treewidth. Our results confirm the observation of Gogate and Dechter [14], 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.

7. Conclusions

In this paper, we have proposed an extended depth-first search (EDFS) algorithm for the optimal triangulation of Bayesian networks. 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, we developed a new algorithm for maintaining the cliques of a dynamic graph. The performance of the proposed algorithm was compared with the state-of-the-art Ottosen and Vomlel [20] and Li and Ueno [24] methods. Our experiments show that the new method is superior to the other methods for graphs with moderate size and low density. Because the moral graphs of Bayesian network are typically moderate in size and sparse, our proposed method provides the best performance for triangulation of Bayesian networks. 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, we proposed the pivot clique pruning theorem in Section 5.2. 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 speedup 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 faster than DFS (with the proposed dynamic clique maintenance) for the repository datasets. The pivot clique pruning also engenders a considerable memory reduction, which is also due to the reduction of the search space.

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 size 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 which obtained an approximate total table size to estimate inference time [26]. Our study on optimal triangulation might improve Mengshoel's results.

Acknowledgements

Parts of this research have been reported in earlier workshop papers [24,29]. The authors would like to thank Dr. T. Ottosen for his helpful comments and discussion in PGM2012 about pivot clique pruning. The authors appreciate the anonymous reviewers' significant and insightful comments.

References

[1] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1988.
[2] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artif. Intell. 42 (2–3) (March 1990) 393–405, http://dx.doi.org/10.1016/0004-3702(90)90060-D.
[3] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1996) 273–302, http://dx.doi.org/10.1016/0004-3702(94)00092-1.
[4] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, J. R. Stat. Soc., Ser. B 50 (2) (1988) 157–224.
[5] F. Jensen, S. Lauritzen, K. Olesen, Bayesian updating in causal probabilistic networks by local computations, CSQ, Comput. Stat. Q. 4 (1990) 269–282.
[6] P.P. Shenoy, G. Shafer, Axioms for probability and belief-function propagation, in: Uncertainty in Artificial Intelligence, Elsevier, Amsterdam, 1990, pp. 169–198.
[7] U. Kjaerulff, Triangulation of graphs – algorithms giving small total state space, Technical Report R9009, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1990.
[8] F.V. Jensen, T.D. Nielsen, Bayesian Networks and Decision Graphs, 2nd edition, Springer-Verlag, 2007.
[9] W. Wen, Optimal decomposition of belief networks, in: Proceedings of the Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-90), Elsevier Science, New York, NY, 1990, pp. 245–256.
[10] A.L. Madsen, F.V. Jensen, Lazy propagation: a junction tree inference algorithm based on lazy evaluation, Artif. Intell. 113 (1) (1999) 203–245.
[11] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.
[12] D.J. Musliner, J.A. Hendler, A.K. Agrawala, E.H. Durfee, J.K. Strosnider, C.J. Paul, The challenges of real-time AI, Computer 28 (1) (1995) 58–66, http://dx.doi.org/10.1109/2.362628.
[13] F.T. Ramos, F.G. Cozman, Anytime anyspace probabilistic inference, Int. J. Approx. Reason. 38 (1) (2005) 53–80, http://dx.doi.org/10.1016/j.ijar.2004.04.001.
[14] V. Gogate, R. Dechter, A complete anytime algorithm for treewidth, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, UAI '04, AUAI Press, Arlington, VA, United States, 2004, pp. 201–208.
[15] E.H. Bachoore, H.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, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 255–266.
[16] P.A. Dow, R.E. Korf, Best-first search for treewidth, in: Proceedings of the 22nd National Conference on Artificial Intelligence, vol. 2, AAAI'07, AAAI Press, 2007, pp. 1146–1151.
[17] H.L. Bodlaender, F.V. Fomin, A.M.C.A. Koster, D. Kratsch, D.M. Thilikos, On exact algorithms for treewidth, ACM Trans. Algorithms 9 (1) (2012) 12, http://dx.doi.org/10.1145/2390176.2390188.
[18] E. Bachoore, H.L. Bodlaender, Weighted treewidth algorithmic techniques and results, in: International Symposium on Algorithms and Computation, Springer, 2007, pp. 893–903.
[19] F. van den Eijkhof, H.L. Bodlaender, M.A. Koster, Safe reduction rules for weighted treewidth, Algorithmica 47 (2) (2007) 139–158.
[20] T. Ottosen, J. Vomlel, All roads lead to Rome: new search methods for the optimal triangulation problem, Int. J. Approx. Reason. 53 (9) (2012) 1350–1366, http://dx.doi.org/10.1016/j.ijar.2012.06.006.
[21] F. Cazals, C. Karande, A note on the problem of reporting maximal cliques, Theor. Comput. Sci. 407 (2008) 564–568.
[22] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, in: 10th Annual International Conference on Computing and Combinatorics (COCOON 2004), Theor. Comput. Sci. 363 (1) (2006) 28–42, http://dx.doi.org/10.1016/j.tcs.2006.06.015.
[23] H.L. Bodlaender, A.M. Koster, F.v.d. Eijkhof, Preprocessing rules for triangulation of probabilistic networks, Comput. Intell. 21 (3) (2005) 286–305, http://dx.doi.org/10.1111/j.1467-8640.2005.00274.x.
[24] C. Li, M. Ueno, A depth-first search algorithm for optimal triangulation of Bayesian network, in: Proceedings of the Sixth European Workshop on Probabilistic Graphical Models, 2012, pp. 187–194.
[25] F.V. Jensen, F. Jensen, Optimal junction trees, in: UAI, 1994, pp. 360–366.

[26] O.J. Mengshoel, Understanding the scalability of Bayesian network inference using clique tree growth curves, Artif. Intell. 174 (2010) 984–1006, http://dx.doi.org/10.1016/j.artint.2010.05.007.
[27] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Ann. Discrete Math., vol. 57, North-Holland Publishing Co., Amsterdam, The Netherlands, 2004.
[28] D.R. Wood, On the maximum number of cliques in a graph, Graphs Comb. 23 (3) (2007) 337–352, http://dx.doi.org/10.1007/s00373-007-0738-8.
[29] C. Li, M. Ueno, A fast clique maintenance algorithm for optimal triangulation of Bayesian networks, in: Proceedings of the Second Workshop on Advanced Methodologies for Bayesian Networks, 2015, pp. 152–167.