A Fast Clique Maintenance Algorithm for Optimal Triangulation of Bayesian Networks

Chao Li(✉) and Maomi Ueno

University of Electro-Communications,
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
{ricyou,ueno}@ai.is.uec.ac.jp

Abstract. The junction tree algorithm is currently the most popular algorithm for exact inference on Bayesian networks. To improve the time and space complexity of the junction tree algorithm, we must find an optimal total table size triangulations. For this purpose, Ottosen and Vomlel proposed a depth-first search (DFS) algorithm for optimal triangulation. They also introduced several techniques for improvement of the DFS algorithm, including dynamic clique maintenance and coalescing map pruning. However, their dynamic clique maintenance might compute some duplicate cliques. In this paper, we propose a new dynamic clique maintenance that only computes the cliques that contain a new edge. The new approach explores less search space and runs faster than the Ottosen and Vomlel method does. Some simulation experiments show that the new dynamic clique maintenance improved the running time of the optimal triangulation algorithm.

Keywords: Optimal triangulation · Junction tree algorithm · Dynamic clique maintenance

1 Introduction

Bayesian networks are graphical models that encode probabilistic relations among variables [1]. A Bayesian network is a directed acyclic graph in which vertices represent random variables, and the arcs represent conditional dependencies. Two vertices that are not connected by an arc represent the two variables that are conditionally independent of each other. Each variable is associated with a conditional probability table conditioning on 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, probabilistic inference using Bayesian network is known to be NP-hard [2]. The network size limitation of the inference algorithm obstructs the more widespread application of Bayesian Networks. Many studies have been undertaken to improve inference algorithms in the past two decades. The most influential exact inference algorithm is the junction tree propagation algorithm [3–5]. In this algorithm, a Bayesian network is first converted into a

J. Suzuki and M. Ueno (Eds.): AMBN 2015, LNAI 9505, pp. 152–167, 2015.
DOI: 10.1007/978-3-319-28379-1_11

special data structure called junction tree; then belief is propagated on that 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 it is necessary to add extra edges to it until it becomes so. This process is called triangulation. A Bayesian network allows several different triangulations. The triangulation is expected to affect the structure of the junction tree and the performance of subsequent belief propagation. In this paper, we especially focus on the optimal triangulation of Bayesian networks. Unfortunately, finding an optimal triangulation is NP-hard [6]. However, this defect is not crucially important because the triangulation can often be done off-line and can be saved for inference algorithms.

Previous investigations of triangulation problems have been conducted by researchers from various fields for different purposes. Their triangulation algorithms are designed to optimize various criteria. The commonly used criteria are the fill-in, the treewidth, and the total table size. Of all these criteria, the total table size criterion yields the most exact bounds of the memory and time requirements of probabilistic inference. Thus, for inference on a Bayesian network, a triangulation is optimal if the triangulation has the minimum total table size. Finding an optimal triangulation is important because the junction algorithm provides the best performance from optimal triangulation. Moreover, optimal triangulation is required for an embedded system that is often with real-time computing constraints and with limited memory usage. We solve the optimal triangulation problem by searching the space of all possible triangulations. This search is conducted by enumerating all possible elimination orders to find the order that has the minimum total table size.

To obtain an optimal triangulation for total table size criterion, Ottosen and Vomlel investigated depth-first search and best-first search algorithms [7]. They claimed that the depth-first search uses less memory than the best-first search does. Moreover, they demonstrated that the two methods have almost equal run time in computational experiments. The best-first search with theoretically better order does not necessarily run faster than the depth-first search in practice. Although the depth-first search expands more search nodes than the best-first search does, the best-first search has heavy overhead costs for maintaining a priority queue (In this paper, the term "node" is used exclusively for a point in the search space for the optimal triangulation algorithm. The term "vertex" is used exclusively for a point in the graph being triangulated.). For optimal triangulation algorithms, it is necessary to make the overhead as low as possible when reducing the search space. To reduce the overhead cost, Ottosen and Vomlel introduced dynamic clique maintenance. In the optimal triangulation algorithm, it is necessary to compute total table size of each search node, which is a lower bound of the node. Therefore, we must also ascertain the set of cliques of each node. It is necessary to maintain a set of cliques in a dynamic graph. The dynamic graph means that the edges can be removed and added but the set of vertices is invariant. To compute the cliques of the updated graph, a simple approach is to run the Bron–Kerbosch (BK) algorithm [8] on the graph. However, the BK algorithm suffers from heavy computational costs. For a graph

with n vertices, the worst-case running time of the BK algorithm is $\mathcal{O}(3^{n/3})$ [9]. To resolve this problem, Ottosen and Vomlel proposed a dynamic clique maintenance [7], which runs the BK algorithm on a smaller subgraph where all the new cliques can be found and all the existing cliques are removed. The dynamic clique maintenance reduced the overhead of each node and made the optimal triangulation algorithm faster. However, the dynamic clique maintenance proposed by Ottosen and Vomlel might compute some duplicate cliques. The method presents shortcomings in computational costs as the number of duplicate cliques becomes large. In the elimination process for triangulating a graph, it is well know that a new fill-in edge cannot connect to the vertex that has been eliminated. Based on this observation, Li and Ueno [10] proposed an improved dynamic clique maintenance. The Li and Ueno method reduced the search range of the BK algorithm by removing eliminated vertices from the graph that the Ottosen and Vomlel method explores. However, the method still computes many duplicated cliques. A new clique in the updated graph must include a new edge. However, these methods might compute some cliques that do not contain a new edge. Those cliques are computed both in the original graph and in the updated graph.

In this paper, we propose a new dynamic clique maintenance algorithm for optimal triangulation of a Bayesian network. When some new edges are inserted in a graph for triangulation purpose, we must update the set of cliques. A new clique in the updated graph must contain a new edge. The idea of our method is to avoid recomputing the cliques that do not contain a new edge. We only explore an even smaller subgraph than the graph that the Ottosen and Vomlel method explores. The subgraph only contains the new edges and their neighboring vertices. We run the BK algorithm on the subgraph where all the new cliques can be found. The new algorithm explores less search space and runs faster than the Ottosen and Vomlel method does. The computational cost of dynamic clique maintenance is inherent in the calculation of the lower bound at each node. Thereby, the improvement of the dynamic clique maintenance algorithm can decrease the overhead of each node. Some simulation experiments show that the new dynamic clique maintenance improved the running time of the optimal triangulation algorithm.

The remainder of this paper is organized as follows. Section 2 presents the triangulation problem and describes the formulation of the search space of the optimal triangulation algorithm. Section 3 reviews the depth-first search algorithm presented in [7]. In Sect. 4, we propose a new dynamic clique maintenance algorithm. Section 5 provides some experiments that are useful to evaluate the proposed method. Section 6 concludes the paper.

2 Triangulation Problem

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

2.1 Notation and Definitions

Let G = (V, E) be an undirected graph with vertex set V and edge set E. For a set of vertices W ⊆ 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 are said to be adjacent if (v, w) ∈ E. The neighbors $\mathcal{N}$(v, G) of a vertex v is the set W ⊆ V such that each u ∈ W is adjacent to v. The family $\mathcal{FA}$(W, G) of a set of vertices W is defined as the set $(\cup_{w \in W} \mathcal{N}(w, G)) \cup W$. Let e = (v,w) ∈ E be an edge, we define the neighbors $\mathcal{N}$(e, G) of an edge e as the set of vertices U ⊆ V such that each u ∈ U is adjacent to v and w. The family $\mathcal{FA}$(F, G) of a set of edges F is defined as the set $(\cup_{f \in F} \mathcal{N}(f, G)) \cup \mathcal{V}$(F). Note the family $\mathcal{FA}$(F, G) of a set of edges F is a subset of the family $\mathcal{FA}(\mathcal{V}$(F), G) of a set of vertices $\mathcal{V}$(F). For example, see the left graph of Fig. 1, we have F = {(a,c),(b,c)}, and $\mathcal{V}$(F) = {a,b,c}, then $\mathcal{FA}$(F, G) = {a,b,c} is a subset of $\mathcal{FA}(\mathcal{V}$(F), G) = V.

A graph G is complete if all pairs of vertices (u,v) (u ≠ v) are adjacent in G. A set of vertices W ⊆ 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 ⊂ U, then W is a *clique*. (Remark: Any complete set is called a clique in some literatures. 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 as $\mathcal{C}$(G). For a set of vertices W ⊆ V, $\mathcal{C}$(W, G) denotes the set of cliques that intersects W. Let G' = (V, E ∪ F) (F ∩ E = ∅) be the right graph obtained by adding a set of new edges F to G = (V, E), $\mathcal{RC}$(G,G') = $\mathcal{C}$(G)\$\mathcal{C}$(G') denotes the set of removed cliques, and $\mathcal{NC}$(G,G') = $\mathcal{C}$(G)\$\mathcal{C}$(G) denotes the set of new cliques. For example, in Fig. 1, let G be the graph on the left, and G' be the graph obtained by adding edge (c,d) to G, then 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}}. Therefore, we have $\mathcal{RC}$(G,G') = {{b,d},{d,e},{c,e}} and $\mathcal{NC}$(G,G') = {{b,c,d},{c,d,e}}.

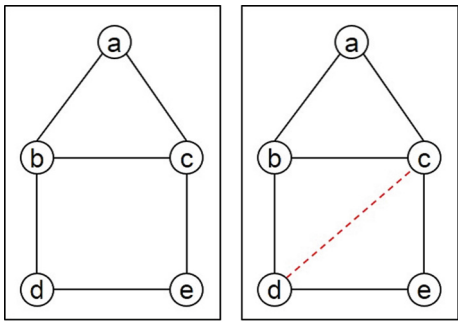


Fig. 1. Left: Initial graph G = (V, E). Right: Updated graph G' obtained by adding one edge (c,d) to G.

The table size of a clique C is defined as $\mathrm{ts(C)} = \prod_{(v \in C)} |sp(v)|$, where $sp(v)$ denotes the state space of the variable corresponding to v in the Bayesian network. The total table size (TTS) of a graph G is defined as $\mathrm{tts(G)} = \sum_{C \in \mathcal{C}(G)} ts(C)$.

A undirected graph G is said to be triangulated if every cycle of length greater than 3 has a chord, that is an edge connecting two nonconsecutive vertices in the cycle. A triangulation of G is defined as a set of edges T such that $\mathrm{T} \cap \mathrm{E} = \emptyset$ and graph $\mathrm{H} = (\mathrm{V}, \mathrm{E} \cup \mathrm{T})$ is triangulated. For example, in Fig. 1, the graph on the left is not triangulated because a chord-less cycle {b,c,e,d} exists. The graph on the right is triangulated because the edge (c, d) is added, which is a triangulation for the graph on the left.

Elimination of a vertex $\mathrm{v} \in \mathrm{V}$ from graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is the process of adding necessary edges F to make the set $\mathcal{N}(\mathrm{v}, \mathrm{G})$ complete, then removing v and all the incident edges from G. The edges F that are added during the elimination process are called fill-in edges. If $\mathrm{F} = \emptyset$, then v is called a simplicial vertex of G. An elimination order for graph G is a total ordering π of the vertices of G, where $\pi(i)$ denotes the i-th vertex in the ordering. Let τ be the partial elimination order, which is a sequence of vertices. Let $\mathcal{V}(\tau)$ denotes the set of vertices presented in τ. Let T be all the fill-in edges that result from eliminating vertices from graph G according to order π. We will then use G^{π} to denote the graph that results from adding these fill-in edges T to $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ and write $G^{\pi} = (\mathrm{V}, \mathrm{E} \cup \mathrm{T})$. Given any elimination order π, if all vertices are eliminated sequentially from G according to π, then the union of all the fill-in edges is a triangulation of G and G^{π} is a triangulated graph.

We present one example for elimination of vertices from a moral graph of Asia [4] Bayesian network in Fig. 2. Consider an elimination order starting with the sequence $\langle D, S \rangle$. Because eliminating vertex D does not add fill-in edge, D is a simplicial vertex. This process induces two associated graphs (filled-in graph and remaining graph). Let $\tau = \langle X \rangle$ denote the partial elimination order, We also refer to the filled-in graph G^{τ} as partially triangulated graph, which is shown in Fig. 2(a). The remaining graph $G^{\tau}[\mathrm{V}/\mathcal{V}(\tau)]$ $(\mathcal{V}(\tau) = \{\mathrm{D}\})$ is shown in Fig. 2(b). Then we eliminate vertex S. Eliminating vertex S adds a fill-in edge (L,B). This process also induces two associated graphs. Let partial order τ' be the vertex sequence $\langle D, S \rangle$, $\mathrm{F} = \{(\mathrm{L,B})\}$ be all fill-in edges when we eliminated along τ'. The corresponding partially triangulated graph $G^{\tau'} = (\mathrm{V}, \mathrm{E} \cup \mathrm{F})$ is shown in Fig. 2(c). The corresponding remaining graph $G^{\tau'}[\mathrm{V}/\mathcal{V}(\tau')]$ $(\mathcal{V}(\tau') = \{\mathrm{D,S}\}$) is shown in Fig. 2(d). If we continue to eliminate vertices until no vertex was left. The final partially triangulated graph (also called filled-in graph) is a correct triangulated graph such that there is no chordless cycle on it. Therefore, triangulation using vertex elimination is simple, but the determination of a good elimination order is the most important step. In this paper, we try to find the order π that induces a triangulated graph with minimum total table size.

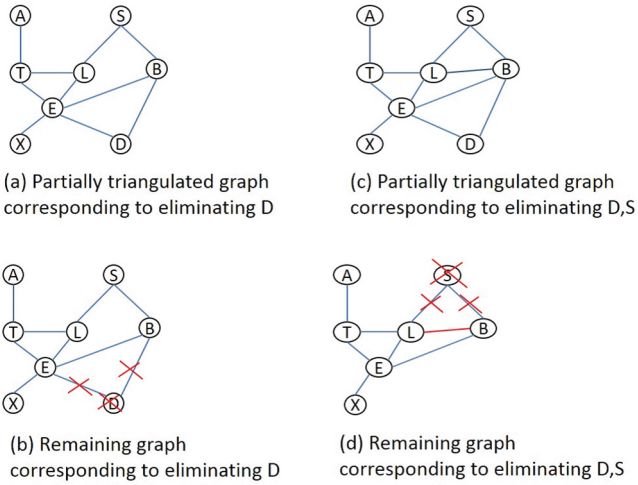


Fig. 2. Example of eliminating vertices from the moral graph of Asia network.

2.2 Search Space of the Optimal Triangulation Algorithm

To find the optimal triangulation, we can conduct a search in the space of all possible elimination orders of the moralized graph of Bayesian network [7]. For this purpose, we generate a search graph that includes all elimination orders for a Bayesian network. Figure 3 depicts the search graph for a network with five vertices. The search graph is a tree with root node corresponding to the start search node and leaf nodes corresponding to all distinct elimination orders. In this search tree, each node is labeled using a partial elimination order τ. We also associate the intermediate partially triangulated graph with each node for reasons of computational convenience in the optimal triangulation algorithm. Each leaf node is labeled using a complete order and is associated with a triangulated graph. For a node labeled τ, the successor node can be generated by the elimination of a vertex from remaining graph $G^{\tau}[V/\mathcal{V}(\tau)]$. Given the search tree, we can explore all possible elimination orders to find the order that has minimum total table size.

3 The Optimal Triangulation Algorithm

This section presents a review of the depth-first search algorithm for optimal triangulation presented by Ottosen and Vomlel [7].

3.1 The Depth-First Search Algorithm for Optimal Triangulation

The naive depth-first branch and bound algorithm for optimal triangulation operates as follows. First, we initialize the upper bound (UB) on total table size

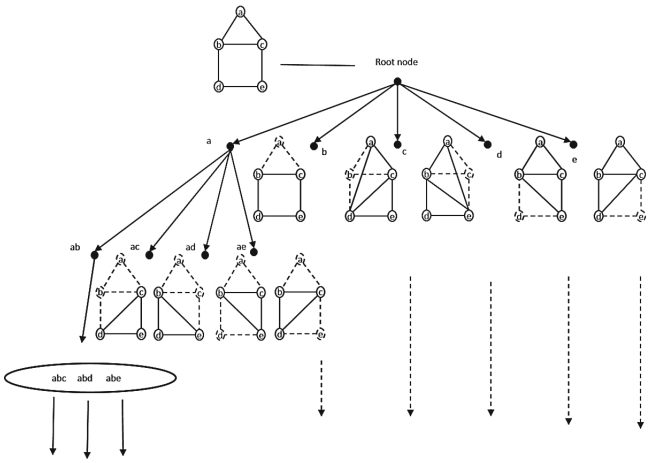


Fig. 3. Search tree of the optimal triangulation algorithm for a graph of five vertices.

(TTS) with the triangulation using minimum fill-in heuristic, which greedily selects the next vertex to eliminate if the vertex leads to add the minimum fill-in edges. Next, it traverses all search tree nodes in a depth-first manner. For each tree node, we calculate the TTS of the partially triangulated corresponding to the node. The TTS is a lower bound of a search node because by adding an edge to graph G, the TTS of G cannot decrease [7]. If we find a node of which TTS is greater than the TTS of UB, then we prune all the successors from the node. On the other hand, if we find a leaf node (labeled using a complete ordering) that is better than UB, we update the UB by replacing the leaf node to UB. The search continues until all nodes have been explored. It is noteworthy that the algorithm performs a search in the space of all elimination orders.

We intend to use the TTS upper bound for pruning nodes in depth-first search triangulations. Therefore, we must compute the TTS of each node in the search tree. The TTS is easy to compute if we know the cliques of the partially triangulated graph corresponding to the node. Therefore, we must also associate the set of cliques with each node. In the Ottosen and Vomlel algorithm [7], for computing the TTS lower bound, each node t is represented as a tuple (τ,H,$\mathcal{C}$,tts,R).

- t.τ: an ordered list of vertices representing the partial elimination order.
- t.H = (V, E $\cup$ F): partially triangulated graph obtained by adding all fill-in edges accumulated along the τ to the original moral graph.
- t.$\mathcal{C}$: A set of cliques for H, $\mathcal{C}$(H).
- t.tts: Total table size of graph H, which is a lower bound for node t.
- t.R: The remaining graph, R = H[V\$\mathcal{V}(\tau)$], where $\mathcal{V}(\tau)$ denotes the set of vertices that lie in τ.

To compute t.tts, we must calculate all the cliques $t.\mathcal{C}$ first. For this purpose, we can use a standard clique enumeration algorithm such as the well-known

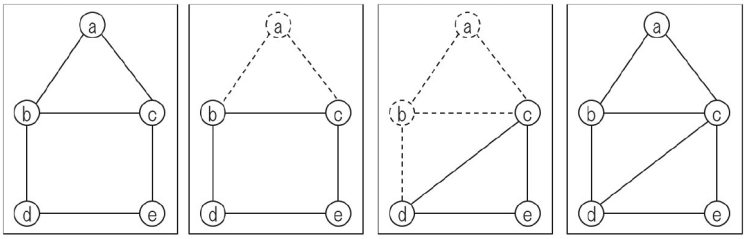


Fig. 4. Example of the vertex elimination and partially triangulated graphs induced by an elimination order that starts with sequence {a,b}. Left: Initial graph. Middle left: Partially triangulated graphs correspond to elimination partial order {a}. Middle right: Partially triangulated graphs correspond to the elimination partial order {a,b}. Right: Final triangulated graph.

Bron–Kerbosch algorithm (BK algorithm) [8]. Below, we present an example to explain the lower bound and related computations.

Example 1. Fig. 4 depicts the vertex elimination process according to the left-most path in Fig. 3. The path corresponds to sequential elimination of vertices a and then b. The *root node* r corresponds to the graph on the left in Fig. 4(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 binary variables) is $3 \cdot 2^2 + 2^3 = 20$, which is a lower estimate of TTS of optimal triangulation.

The successor node t of r (induced by elimination of vertex a) corresponds to the graph on the middle-left in Fig. 4. The partially triangulated graph t.H is the same as the initial one. Therefore, we can derive $t.tts = 20$.

We expand the successor node t' of t (corresponding to the elimination of vertex b). The induced partially triangulated graph t'.H corresponds to the middle-right graph in Fig. 4, which includes the fill-in edge (c, d). This process continues until the graph is triangulated. The resulting triangulated graph corresponds to the right graph in Fig. 4, which is the same as t'.H. Finally, the cliques of the triangulated graph are $t'.\mathcal{C} = \{\{a,b,c\},\{b,c,d\},\{c,d,e\}\}$. Their $t'.tts$ 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 makes sure the correctness of applying branch and bound technique in the optimal triangulation algorithm.

However, the BK algorithm suffers from heavy computational cost. For a graph with n vertices, the worst-case running time of the BK algorithm is $\mathcal{O}(3^{n/3})$. Indeed, the BK algorithm engenders many redundant computations. To tackle this problem, Ottosen and Vomlel [7] proposed a more efficient algorithm for computation of the set of cliques $\mathcal{C}$ in a dynamic graph. We will explain the dynamic clique maintenance algorithm in Sect. 3.2.

We explained the search tree of depth-first search and how to compute a lower bound for each node. The depth-first search algorithm presented by Ottosen and

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:     Let best = MinFill(s)                           ▷ Best path
 7:     Let map = ∅                                     ▷ Coalescing map
 8:     ExpandNode(s,best,map)                          ▷ Start recursive call return best
 9: procedure ExpandNode(n,&best,&map)
10:     for all v ∈ V(n.R) do
11:         Let m = Copy(n)
12:         EliminateVertex(m, v)                       ▷ Update graph, cliques and TTS
13:         EliminateSimplicial(m)                      ▷ Simplicial vertex rule
14:         if |V(m.R)| = 0 then
15:             if m.tts<best.tts then
16:                 Set best = m
17:         else
18:             if m.tts≥best.tts then
19:                 continue                            ▷ Branch and bound
20:             if map[m.R].tts≤m.tts then
21:                 continue
22:             Set map[m.R] = m
23:             ExpandNode(m,best,map)
```

Vomlel can be implemented in O(|V|) space and O(|V|!) time. A pseudo code of the Ottosen and Vomlel algorithm is shown in Algorithm 1. The Eliminate Vertex(m,v) procedure eliminates vertex v from the remaining graph of node m. To prune unnecessary search nodes further, Ottosen and Vomlel also introduced the following pruning rules: (1) Graph reduction techniques called the simplicial vertex rule [11,12], and (2) pruning based on a coalescing map. The procedure EliminateSimplicial(m,v) sequentially removes all simplicial vertices from the remaining graph of node m. Coalescing map uses $\mathcal{O}(n^2)$ memory space to prune unnecessary search nodes; see [13] for details. Although it is well known that the depth-first search runs in O(|V|!) time, the algorithm combined with these techniques described above merely hits the upper bound. Ottosen and Vomlel [7] claimed that their algorithm runs in $\mathcal{O}(2^{|V|})$ time in practice.

3.2 Previous Works on Dynamic Clique Maintenance

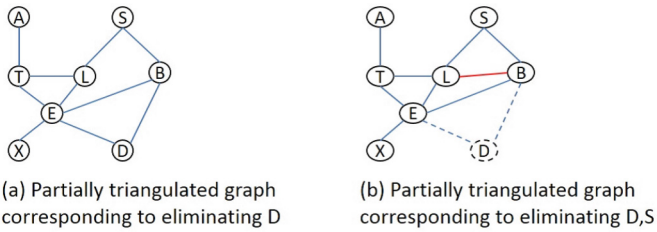
Ottosen and Vomlel [7] observed that recomputing all cliques of a graph using the BK algorithm is unnecessary. Then they proposed the following dynamic clique maintenance algorithm. The main idea behind the algorithm is the following. Instead of searching for all cliques in the whole graph, as the BK algorithm does, their algorithm runs a clique enumeration algorithm simply on a smaller subgraph on which all the new cliques can be found and all the existing cliques are

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

```
1: procedure CliqueUpdate(G, 𝒞(G), F)
2:     Let G' = (V, E ∪ F)
3:     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:     let C^new = BKalgorithm(G'[ℱ𝒜(U, G')])
9:     for each clique C ∈ C^new do                   ▷ Add new cliques
10:        if C ∩ U ≠ ∅ then
11:            Set 𝒞(G') = 𝒞(G') ∪ {C}
```

removed. 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 Ottosen and Vomlel algorithm is derived based on the following theorem:

Theorem 1 ([7]). *Let $G=(V, E)$ be an undirected graph, and let $G'=(V, E\cup F)$ be the graph result from adding a set of new edges F to G. Let $U=\mathcal{V}(F)$, and let $\mathcal{C}(G')=\mathcal{C}(G)$. We remove the cliques of $\mathcal{C}(U,G)$ from $\mathcal{C}(G')$ and add cliques of $\mathcal{C}(U,\mathcal{FA}(U, G'))$ to $\mathcal{C}(G')$,then $\mathcal{C}(G')$ is the set of all cliques of G'.*



(a) Partially triangulated graph corresponding to eliminating D

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

Fig. 5. A sequence of graphs corresponding to eliminating of vertices D and S. (L,B) is the fill-in edge.

Next, we provide an example to trace Algorithm 2.

Example 2. Consider the left graph G in Fig. 5. $\mathcal{C}(G)$ 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 right graph in Fig. 5). The set U = {L,B} and we let $\mathcal{C}(G')=\mathcal{C}(G)$.

First, we iterate 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 the $\mathcal{C}$(G') = {{A,T}, {E,X},{T,L,E}, {S,L,B}, {L,B,E}, {B,D,E}}, which is the cliques of new graph G'.

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

```
1: procedure CliqueUpdate1(G, C(G), F, W)
2:     Let G' = (V, E ∪ F)
3:     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:     let C^new = BKalgorithm(G'[FA(U, G')\W])
10:    for each clique C ∈ C^new do                         ▷ Add new cliques
11:        if C ∩ U ≠ ∅ then
12:            Set C(G') = C(G') ∪ {C}
```

The example shows that the algorithm sometimes removes and adds the same cliques again. Although the Ottosen and Vomlel method reduces the search range of the BK algorithm from the whole graph to a small subgraph G'[$\mathcal{FA}$(U, G')], the method might present shortcomings in performance when the number of duplicated 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 the vertex that has been eliminated. The neighbors of D are invariant in G and G'. Therefore, 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 [10] proposed an improved dynamic clique maintenance. The improved dynamic clique maintenance 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 BK algorithm on the graph G'[$\mathcal{FA}$(U, G')\W], which is a subgraph of G'[$\mathcal{FA}$(U, G')] which the Ottosen and Vomlel method explores. As the BK algorithm is the most time consuming part in the dynamic clique maintenance procedure. For a graph with n vertices, the worst-case running time of the BK algorithm is $\mathcal{O}(3^{n/3})$. Therefore, this reduction of the search range is important to improve the performance of the 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 is disjoint W. This check is usually not a problem because the complexity of comparison of cliques is constant if we store a clique using BitSet Object in JAVA programming language.

4 Proposed Dynamic Clique Maintenance

In the depth-first search triangulation algorithm, it is necessary to compute the TTS lower bound of each search node. Therefore, the computational cost of dynamic clique maintenance is inherent in calculation of the lower bound at each node. To lower the overhead cost in the triangulation algorithm, we must compute the cliques of each graph efficiently.

Algorithm 4. Proposed dynamic clique maintenance.

```
1: procedure CliqueUpdate2(G, 𝒞(G), F)
2:     Let G' = (V, E ∪ F)
3:     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:     𝒞(G') = 𝒞(G') ∪ BKalgorithm(G'[W])              ▷ Add new cliques
```

In Sect. 3.2, we have demonstrated by example that the Ottosen and Vomlel approach might compute some duplicated clique. To resolve this problem, we propose a new dynamic clique maintenance algorithm. The main idea of our method is to avoid recomputing the cliques that do not contain a new edge. When some new edges are inserted to a graph, a new clique must contain a new edge. We find that all new cliques and removed cliques are included in the vertex set $\mathcal{FA}(F, G')$, where F is the set of new edges. Therefore, we can only run the BK algorithm on the subgraph $G[\mathcal{FA}(F, G')]$. The proposed dynamic clique maintenance algorithm is shown in Algorithm 4, where U, G, F, $\mathcal{C}(G)$ are defined in the same manner as presented in Algorithm 2, and $W = G[\mathcal{FA}(F, G')]$ 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 $U = \mathcal{V}(F)$, and let $\mathcal{C}(G') = \mathcal{C}(G)$. We remove all the cliques included in $\mathcal{FA}(F, G')$ from $\mathcal{C}(G')$ and add cliques of $\mathcal{C}(G'[\mathcal{FA}(F, G')])$ to $\mathcal{C}(G')$, then $\mathcal{C}(G')$ is the set of all cliques of G'.*

Proof. If a clique C is a new clique in G', $C \in \mathcal{NC}(G,G')$, then C must contain at least one new edge $f \in F$; otherwise C is not a new clique. Because C contains a new edge, any vertex $v \in C$ must be a neighbor of one of the new edges. For any vertex $v \in C$, v must in the set $\mathcal{FA}(F, G')$. Therefore, $C \subseteq \mathcal{FA}(F, G')$. That is to say, all the new cliques can be found on the subgraph $G[\mathcal{FA}(F, G')]$.

If a clique C is a removed clique, $C \in \mathcal{RC}(G,G')$, then there exists a new clique K such that $C \subseteq K$. Because the only way we remove a clique is by replacing the old clique by a new clique K such that $C \subseteq K$. From the result presented above,

a new clique $K \subseteq \mathcal{FA}(F, G')$. Therefore, each removed clique C 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 on the subgraph $G[\mathcal{FA}(F, G')]$ to $\mathcal{C}(G')$. Then, $\mathcal{C}(G')$ is the set of all cliques of G'.

The following example explains the algorithm:

Example 3. Consider the graph G and updated graph G' in Fig. 5. $\mathcal{C}(G)$ is the set of cliques of G, {{A,T}, {T,L,E}, {E,X}, {S,L}, {S,B}, {B,D,E}}. We let $\mathcal{C}(G') = \mathcal{C}(G)$. We first compute a family of edge set F, $W = \mathcal{FA}(F, G')$, $W = \{S,E,L,B\}$. Next, we remove from $\mathcal{C}(G')$ all the cliques that are 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 $\mathcal{FA}(\mathcal{V}(F), G')$ that is used in Ottosen and Vomlel algorithm. Our method makes the BK algorithm explore less search space for updating cliques than the Ottosen and Vomlel method does. The BK algorithm is the most time-consuming part for the dynamic clique maintenance. Therefore, this reduction of the search range is important to improve the performance of the dynamic clique maintenance.

Finally, we simply add all new cliques C^{new} to $\mathcal{C}(G')$. In the Ottosen and Vomlel approach, it is necessary to check each clique in $G'[\mathcal{FA}(U, G')]$ to ascertain whether it intersects with U, or not. However, we relax this conditional check in our algorithm. In this example, we only remove cliques $\mathcal{RC}(G,G')$ and add cliques $\mathcal{NC}(G,G')$. On the other hand, the Ottosen and Vomlel method removes some duplicated cliques and adds those again.

The new dynamic clique maintenance algorithm performs the BK algorithm on $W = G[\mathcal{FA}(F, G')]$, which is a subgraph of $G'[\mathcal{FA}(U, G')]$ on which the Ottosen and Vomlel method does. The clique enumeration algorithm (BK algorithm) entails exponential costs with the number of vertices. In the dynamic clique maintenance algorithm, the running of BK algorithm is the most time consuming part. Therefore, the reduction of search range is effective to reduce calculation costs of dynamic clique maintenance. In the Ottosen and Vomlel approach, a newly found clique has to be check whether it intersects U, or not. Our method relaxes this conditional check and simply adds the all new cliques found in W. To conclude, the new algorithm explores less search space and runs faster than the Ottosen and Vomlel method does. We demonstrate the performance superiority of the new algorithm by simulation experiments in Sect. 6.

5 Experiments

We conducted computational experiments to evaluate the performance of our proposed dynamic clique maintenance. We also compared our algorithm with the

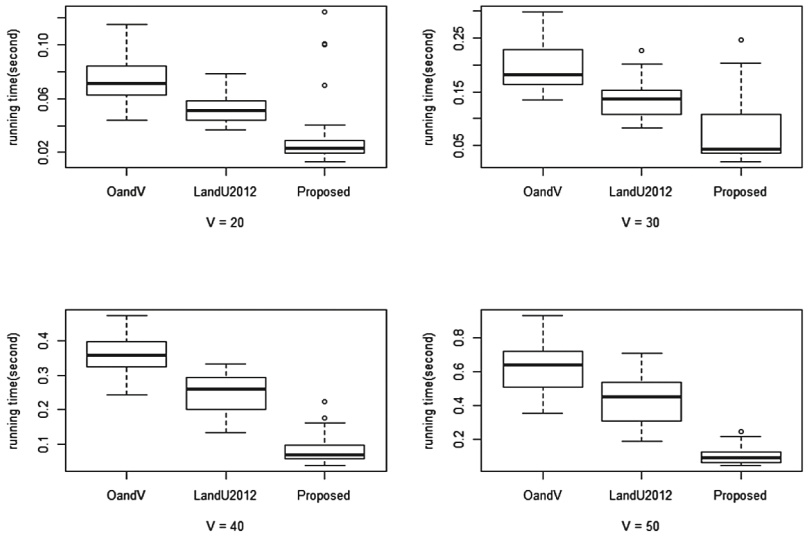


Fig. 6. Comparison of the running times of the Ottosen and Vomlel method (OandV), the Li and Ueno method (LandU2012) and the proposed method for dynamic clique maintenance.

Ottosen and Vomlel method (OandV)[7], and the Li and Ueno (LandU2012)[10]. All algorithms described in this paper were implemented in the Java language in the same manner. All experiments were conducted on a 3.0 GHz processor (Xeon-5675; Intel Corp.) with 12 GB of RAM.

5.1 Dynamic Clique Maintenance

This section presents a comparison of several dynamic clique maintenance methods. For this purpose, we generated 40 random Bayesian networks each for 20, 30, 40, and 50 vertices with various density using BNGenerator[1]. For each graph in the dataset, we triangulated the graph 1,000 times by sequentially eliminating all vertices (with different random elimination order on each run). The set of cliques of the graph is updated after each vertex is eliminated. We chose this experimental scenario because it shows the expected speedup of our proposed method for the triangulation problem. Figure 6 shows the total running time of 1,000 times triangulation for all random Bayesian networks in the dataset. It is clear that the proposed dynamic clique maintenance is faster than OandV and LandU2012.

5.2 Optimal Triangulation

This section describes the experimentally obtained results for the optimal triangulation algorithm for total table size criteria. To examine the effectiveness of our

[1] http://sites.poli.usp.br/pmr/ltd/Software/BNGenerator/.

Table 1. Comparison of depth-first search triangulation algorithms among various dynamic clique maintenance methods.

Bayesian networks		DFS	DFS-1	DFS-2
Name	V	Time (s)	Time (s)	Time (s)
Insurance	27	1.656	1.134	0.776
Water	32	10.415	6.247	4.767
Mildew	35	13.285	8.193	5.246
Alarm	37	0.013	0.010	0.006
Hailfinder	56	8.869	7.493	4.337
WIN95PTS	76	60.082	44.122	27.146

proposed dynamic clique maintenance in the optimal triangulation algorithms, we implemented the following algorithm:

DFS: depth-first search (DFS) optimal triangulation algorithm obtained by introducing OandV dynamic clique maintenance.

DFS-1: improved DFS obtained by introducing LandU2012 dynamic clique maintenance.

DFS-2: improved DFS obtained by introducing the proposed dynamic clique maintenance.

We used six well-known graphs in the Bayesian network repository[2]. The running times of optimal triangulation algorithms are presented in Table 1. Results show that our proposed dynamic clique maintenance dramatically improves the running time of triangulation. This result suggests that the proposed method can extend the available network size of Bayesian network inference.

6 Conclusion

In this paper, we proposed a fast clique maintenance algorithm for optimal triangulation of Bayesian Networks. The performance of the proposed algorithm was compared with the state-of-the-art, the Ottosen and Vomlel method, and the Li and Ueno method. Theoretically analysis and experiments reveal that the new method is superior to the previous proposed method.

Given graph G, new edges F, and eliminated vertex set W, consider the problem of updating cliques of new graph. The Ottosen and Vomlel method runs BK algorithm on G'[$\mathcal{FA}(\mathcal{V}(F)$, G')]. The Li and Ueno method runs BK algorithm on G'[$\mathcal{FA}(\mathcal{V}(F)$, G')\W]. The proposed method runs BK algorithm on G'[$\mathcal{FA}$(F, G')]. The proposed method is faster than the Li and Ueno method. The Li and Ueno method is faster than the Ottosen and Vomlel method. The main reason for the results is that the BK algorithm suffers from heavy computational cost and the proposed method reduces search space for BK algorithm because G'[$\mathcal{FA}(\mathcal{V}(F)$, G')] $\supseteq$ G'[$\mathcal{FA}(\mathcal{V}(F)$, G')\W] $\supseteq$ $G'[\mathcal{FA}(F, G')]$.

[2] http://compbio.cs.huji.ac.il/Repository/networks.html.

The Li and Ueno method (2012) assumes eliminating processes in which some edges are added in a step-by-step manner. Application of the method in other areas such as protein interaction network is expected to create a problem. However, the proposed method does not assume this eliminating process. The proposed dynamic clique maintenance is more generally applicable.

References

1. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco (1988)
2. Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian belief networks. Artif. Intell. **42**, 393–405 (1990)
3. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Roy. Stat. Soc. Ser. B (Methodol.) **50**, 157–224 (1988)
4. Jensen, F., Lauritzen, S., Olesen, K.: Bayesian updating in causal probabilistic networks by local computations. Computat. Stat. Q. **4**, 269–282 (1990)
5. Shenoy, P.P., Shafer, G.: Axioms for probability and belief-function propagation. In: Uncertainty in Artificial Intelligence. Elsevier, Amsterdam (1990)
6. Wen, W.: Optimal decomposition of belief networks. In: Proceedings of the Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI 1990), pp. 245–256. Elsevier Science, New York (1990)
7. Ottosen, T., Vomlel, J.: All roads lead to rome: new search methods for the optimal triangulation problem. Int. J. Approx. Reason. **53**, 1350–1366 (2012)
8. Cazals, F., Karande, C.: A note on the problem of reporting maximal cliques. Theor. Comput. Sci. **407**, 564–568 (2008)
9. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques. In: Chwa, K.-Y., Munro, J.I. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 161–170. Springer, Heidelberg (2004)
10. Chao, L., Maomi, U.: A depth-first search algorithm for optimal triangulation of Bayesian network. In: Proceedings of the Sixth European Workshop on Probabilistic Graphical Models (2012)
11. Bodlaender, H.L., Koster, A.M.C.A., van den Eijkhof, F.: Preprocessing rules for triangulation of probabilistic networks. Comput. Intell. **21**, 286–305 (2005)
12. van den Eijkhof, F., Bodlaender, H.L., Koster, A.M.C.A.: Safe reduction rules for weighted treewidth. Algorithmica **47**, 139–158 (2007)
13. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press, Cambridge (2009)