A Depth-First Search Algorithm for Optimal Triangulation of Bayesian Network

Chao Li
University of Electro-Communications, Japan
ricyou@ai.is.uec.ac.jp

Maomi Ueno
University of Electro-Communications, Japan
ueno@ai.is.uec.ac.jp

Abstract

Finding the triangulation of a Bayesian network with minimum total table size reduces the computational cost for probabilistic reasoning in the Bayesian network. This task can be done by conducting a search in the space of all possible elimination orders of the Bayesian network. However, such a search is known to be NP-hard. To relax this problem, Ottosen and Vomlel (2010b) proposed a depth-first branch and bound algorithm, which reduces the computational complexity from $\Theta(\beta \cdot |\mathrm{V}|!)$ to $\mathrm{O}(\beta \cdot |\mathrm{V}|!)$, where β describes the overhead computation per node in the search space, and where $|\mathrm{V}|!$ is the search space size. Nevertheless, this algorithm entails a heavy computational cost. To mitigate this problem, this paper presents a proposal of an extended algorithm with the following features: (1) Reduction of the search space to O ((|V|-1)!) using Dirac's theorem, and (2) reduction of the computational cost β per node. Some simulation experiments show that the proposed method is consistently faster than Ottosen and Vomlel's method.

1 Introduction

The most influential method for exact inference in Bayesian networks is the join tree propagation algorithm (Jensen et al., 1990), which works in two steps: compilation and propagation. The compilation part of the method

- triangulates the graph (i.e., add extra fill-in edges such that every cycle of length greater than three has a chord),
- forms a factor Φ_C for each clique C of the triangulated graph, and
- constructs a junction tree over these cliques.

The propagation part of the method passes messages in the whole junction tree. In the Hugin architecture, the message passing can be organized so that one passing in each direction of the links of the junction tree. The computational cost for each message computation is proportional to the factor table size of clique. To reduce the computational cost, we need to find a triangulation with minimum total table size, which is the summation of the factor table sizes of all cliques.

Heuristic algorithms include popular triangulation heuristics like min-degree, and min-fill is wildly used for triangulation (Darwiche, 2009), but these heuristics do not guarantee the exact solution.

Although finding the optimal triangulation can be conducted by searching all possible elimination orders, the problem is known to be NP-hard (Wen, 1990). Fortunately, this is not a critical defect because triangulation can often be done off-line on specialized servers. Moreover, the triangulation results can be saved for inference algorithms. To tackle the optimal triangulation, Gogate and Dechter (2004) used

depth-first search and Dow and Korf (2007) used best-first search. However, they used treewidth criterion, which can not guarantee the minimum TTS.

To obtain the exact solution for TTS criterion, Ottosen and Vomlel (2010b) proposed a branch and bound algorithm. This algorithm uses TTS of the current partially triangulated graph as a lower bound. To compute this lower bound for each node on the search space, the cliques of the current graph must be calculated. For this purpose, Ottosen and Vomlel (2010a) proposed a dynamic clique maintenance algorithm. Consequently, they reduced the computational complexity from $\Theta(\beta \cdot |V|!)$ to $O(\beta \cdot |V|!)$, where β describes the per-node overhead computation and where $|V|!$ is the size of the search space. However, this method still entails a heavy computational cost.

To relax this problem, based on Ottosen and Vomlel (2010b), this paper presents a proposal of a depth-first search with the following features:

1. Reduction of search space using Dirac's theorem to O((|V|-1)!), and

2. reduction of the computational cost β per node.

Some simulation experiments show that the proposed method is consistently faster than Ottosen and Vomlel's method.

2 Preliminaries

We shall use the following definitions and notations according to Ottosen and Vomlel (2010b). G = (V, E) is an undirected graph with vertices V=$\mathcal{V}$(G) and edges E=$\mathcal{E}$(G). For a set of edges F, $\mathcal{V}$(F) is the set of vertices {u,v | (u,v) ∈ F}. For a set of vertices W ⊆ V, G[W] is the subgraph of G that is induced by W.

Two vertices u and v are connected in G if an edge exists between them. A graph G is complete if all pairs of vertices {u,v}(u≠v) are connected in G. A set of vertices W⊆V is complete in G if G[W] is a complete graph. If W is complete and no complete set U exists such that W ⊂ U, then W is a *clique*. (Remark: Any complete set is sometimes called a clique. Therefore, what we defined as a clique is called a maximal clique.) The set of all cliques of a graph is denoted as $\mathcal{C}$(G). The set of all cliques that intersect a set of vertices W is denoted as $\mathcal{C}$(W, G).

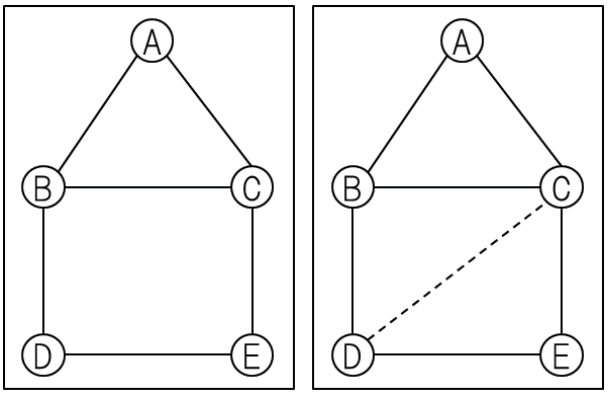

Figure 1: Left: Initial graph G = (V,E). Right: Updated graph G' by adding one edge {C,D}. We have $\mathcal{C}$(G) = {{A,B,C},{B,D},{D,E},{C,E}} and $\mathcal{C}$(G') = {{A,B,C},{B,C,D},{C,D,E}}. In this example, we have $\mathcal{RC}$(G,G') = {{B,D},{D,E},{C,E}} and $\mathcal{NC}$(G,G') = {{B,C,D},{C,D,E}}.

The neighbors nb(v,G) of a vertex v ∈ V is the set W ⊂ V such that each u ∈ W is connected to v. The neighbors nb(W,G) of a set of vertices W are those vertices from V\W, which are connected to at least one vertex v ∈ W. The *family* fa(W,G) of a set of vertices W is the set nb(W,G)∪W.

If G' = (V,E∪F) is the graph resulting from adding a set of new edges F to G, then $\mathcal{RC}$(G,G') = $\mathcal{C}$(G) \ $\mathcal{C}$(G') is the set of removed cliques, and $\mathcal{NC}$(G,G') = $\mathcal{C}$(G')\ $\mathcal{C}$(G) is the set of new cliques. Figure 1 presents these concepts.

An undirected graph is *triangulated* (or chordal) if every cycle of length four or more has a chord that is an edge of the graph joining two non-adjacent vertices of the cycle. A *triangulation* of G is a set of edges T such that T ∩ E= ∅ and the graph H = (V,E ∪ T) is triangulated. We denote the set of all triangulations of a graph G for $\mathcal{T}$(G). For example, in Figure 1, the graph on the left is not triangulated because there exists a cycle {B,C,D,E} of length four in which a chord does not exist. The graph on the

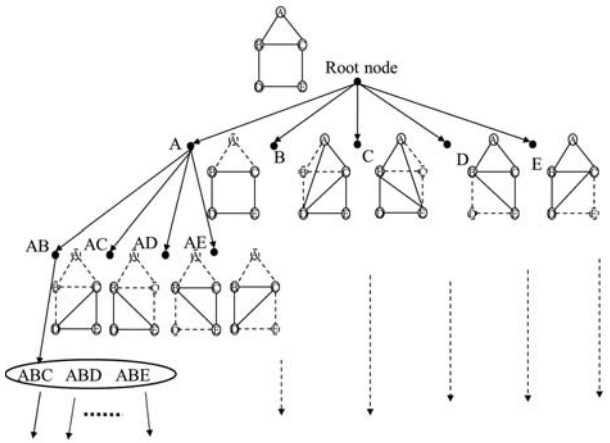

Figure 2: Search tree for exploring all elimination orders of the graph.

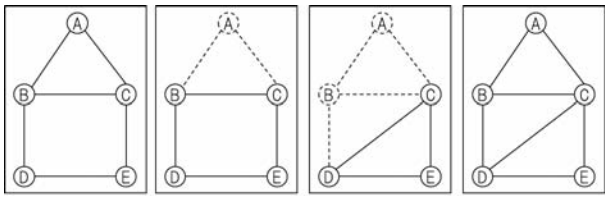

Figure 3: Example of the vertex elimination and partially triangulated graphs induced by an elimination order that starts with sequence {A,B}. Left: Initial graph. Middle left: Partially triangulated graphs correspond to elimination order prefix {A}. Middle right: Partially triangulated graphs correspond to elimination order prefix {A,B}. Right: Final triangulated graph.

right is triangulated because the edges {{C,D}} is a triangulation T for the graph on the left.

The elimination of a vertex v ∈ V of G = (V,E) is the process of removing v from G and making nb(v,G) a complete set. This process induces a partially triangulated graph H = (V, E∪F), where F is the set of fill-in edges. If F = ∅, then v is a simplicial vertex. An *elimination order* $\pi = \{v_1, v_2, \ldots, v_n\}$ is a permutation of vertices specifying an ordering for eliminating all vertices V of G. A prefix τ of an elimination order π is a sequence of vertices that occurs at the beginning of π. If all vertices are eliminated in G according to an elimination order π, then the union of all the fill-in edges induces a triangulation T of G. Consequently, each triangulation T of G corresponds to at least one elimination order. We can explore the space $\mathcal{T}$(G) by investigating all possible elimination orders.

The table size of a clique C is given as 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. Finally, the TTS of a graph H is given as tts(H) = $\sum_{C \in \mathcal{C}(H)} ts(C)$.

3 Previous works

3.1 Depth-first branch and bound algorithm for triangulation

To find the optimal triangulation, we investigate all possible elimination orders. Branch and bound is a general efficient algorithm for finding the optimal solution of a triangulation problem. To employ this algorithm, we must branch a search tree and compute a lower bound for each node.

First, we generate the search tree as follows. Each node corresponds to an elimination order prefix τ. Each edge corresponds to a particular vertex being eliminated. Herein, we use the term "node" exclusively for a point in the search tree, and the term "vertex" exclusively for a point in the graph being triangulated. The exploration begins with *root node* r on the search tree, which corresponds to no vertex having been eliminated from G. The goal nodes corresponding to all vertices have been eliminated. Figure 2 depicts an example of a search tree.

To compute a lower bound for each node in the search tree, we must calculate the following with each node t:

- t.H = (V,E∪F): Original graph with all fill-in edges F accumulated according to the elimination order prefix t.τ.
- t.R: Remaining vertices V\W, where W are the all vertices of τ.
- t.$\mathcal{C}$: A set of cliques for H.
- t.tts: Total table size for the graph H, which is a lower bound for node t.

To compute t.tts, we must calculate all the cliques $t.\mathcal{C}$. For this purpose, we can use a standard algorithm such as the well-known Bron–Kerbosch algorithm (BK algorithm) (Cazals and Karande, 2008). The following example

shows how we explore the leftmost path in Figure 2.

Example 1. Figure 3 depicts the vertex elimination process according to the leftmost path in Figure 2. The path follows the elimination order prefix {A,B}. The *root node* r corresponds to the graph on the left (initial graph), where no vertex has been eliminated. We can compute $r.\mathcal{C}$ = {{A,B,C},{B,D},{D,E},{C,E}} using the BK algorithm. The TTS (assuming binary variables) is $3\cdot 2^2+2^3 = 20$, which is a lower estimate of the TTS of the triangulated graph (Ottosen and Vomlel, 2010b).

The successor node t of r (corresponding to the elimination of vertex A) corresponds to the graph on the middle-left. The graph is the same as the initial one, so we can derive tts=20.

Then we explore successor note t' of t (corresponding to the elimination of vertex B). Therefore, the relevant graph corresponds to middle-right including the fill-in edge {C, D}. This process continues until the graph is triangulated. Finally, the cliques of the triangulated graph are $\mathcal{C}$ = {{A,B,C},{B,C,D},{C,D,E}}, and their TTS is $3\cdot 2^3 = 24$.

We explained the search tree and how to compute a lower bound for each node. The branch and bound algorithm for optimal triangulation can be implemented in O(|V|) space and O(|V|!) time.

Ottosen and Vomlel (2010b) improved the naive branch and bound algorithm by adding the following pruning rules: (1) Graph reduction techniques called the simplicial vertex rule (Bodlaender et al., 2005), and (2) pruning based on a coalescing map, by using the fact that the remaining graph H[V\W] is the same irrespective of the order the vertices in W have been eliminated(Dow and Korf, 2007). Although the Branch and Bound theoretically runs in O(|V|!), it merely hits the upper bound. Ottosen and Vomlel (2010b) pointed out that their Branch and Bound with coalescing of nodes actually runs in $O(2^{|V|})$ time.

Algorithm 1 shows Ottosen's algorithm. The algorithm operates as follows: We initialize the best solution with the minimum fill-in heuristic, set best.tts as the upper bound, and then start a branch and bound algorithm to seek a

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)
4:      if |V(s.R)|=0 then
5:          return s
6:      Let best=MinFill(s)        ▷ upper bound
7:      map = ∅                    ▷ Coalescing map
8:      ExpandNode(s,best,map) 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)
13:         EliminateSimplicial(m)
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
20:             if map[m.R].tts≤m.tts then
21:                 continue
22:             Set map[m.R] = m
23:             ExpandNode(m,best,map)
```

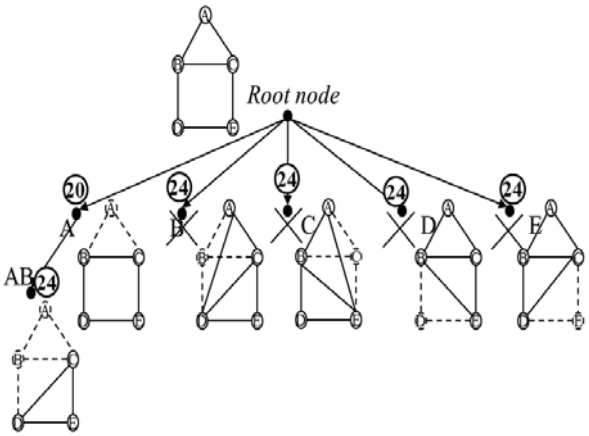

Figure 4: Example of the upper bound pruning. The number in circle is TTS of node. We first generate a suboptimal solution according to the minimum fill-in heuristic, which is the elimination order beginning from A,B then use this upper bound 24 to prune the other nodes of which TTSs are more than or equal to this upper bound.

Algorithm 2 maintenance of cliques by local search

```
1:  procedure UPDATE(G,&C,tts,F)
2:      set G'=(V,E∪F)
3:      Let U = V(F)
4:      for all C ∈ C(G) do
5:          if C ∩ U ≠ ∅ then
6:              Set tts = tts -ts(C)
7:              Set C = C \C
8:      let C^new = FindCliques(G[fa(U,G')])
9:      for all C ∈ C^new do
10:         if C ∩ U ≠ ∅ then
11:             Set tts = tts + ts(C)
12:             Set C = C ∪ C
```

Algorithm 3 Clique maintenance with reduced search.

```
1:  procedure UPDATE(G,&C,tts,F,W)
2:      set G'=(V,E ∪ F)
3:      Let U = V(F)
4:      for all C ∈ C(G) do
5:          if C ∩ U ≠ ∅ then
6:              if C ∩ W = ∅ then
7:                  Set tts = tts -ts(C)
8:                  Set C = C \C
9:      let C^new = FindCliques(G[fa(U,G')\W])
10:     for all C ∈ C^new do
11:         if C ∩ U ≠ ∅ then
12:             Set tts = tts + ts(C)
13:             Set C = C ∪ C
```

better solution. When we find a node of which the lower bound of TTS is greater than best.tts, we prune the node. Figure 4 describes an example of the pruning procedure. If we find a better ordering, we update the best solution. The procedure EliminateVertex $(\cdot)$ simply eliminates a vertex from the remaining graph R and updates the cliques and TTS of the partially triangulated graph. The procedure EliminateSimplicial $(\cdot)$ removes all simplicial vertices from the remaining graph.

3.2 Dynamic clique maintenance

To compute t.$\mathcal{C}$ for each node t, we can use a standard algorithm like the BK algorithm, which compute all cliques of a graph. Ottosen and Vomlel (2010b) pointed out that recomputing all cliques is unnecessary and proposed the following dynamic clique maintenance algorithm.

The general idea behind the algorithm is simple. Instead of searching for all cliques in the whole graph, simply run a clique enumeration algorithm on a smaller subgraph where all the new cliques appear and existing cliques disappear. This dynamic clique maintenance method is presented in Algorithm 2. As a side benefit, it also updates the TTS and the current graph. Algorithm 2 is derived based on the following theorem:

Theorem 1 (Ottosen and Vomlel,2010a)**.** *Let G = (V,E) be an undirected graph, and let G' = (V,E ∪ F) be the graph resulting from adding a set of new edges F to G. Let U = $\mathcal{V}$(F). The $\mathcal{C}$(G') can be found by removing the cliques from $\mathcal{C}$(G) that intersect with U and adding cliques of G'[fa(U,G')] that intersect with U.*

Example 2. Consider the middle-right graph in Figure 3. We update the graph G = (V,E) on the middle-left with the set of fill-in edges F ={{C,D}}(line 2 in Algorithm 2). The set U = {C,D} and fa(U,G') = {A,B,C,D,E}. We iterate through the existing cliques and remove those that intersect with U (lines 4–7), and then add all the cliques found in the subgraph G'[fa(U,G')] (lines 8–12) but which also intersect with U. We can observe that $\mathcal{RC}$(G,G') ∪ {A,B,C} are removed and that $\mathcal{NC}$(G,G') ∪ {A,B,C} are added. The clique {A,B,C} is removed and added again using this algorithm.

The example shows that the algorithm sometimes removes and adds the same cliques again. Although Ottosen's approach reduces the search range of the expensive enumeration algorithm FindCliques$(\cdot)$ to a small subgraph, the method might have a potential performance problem such as duplicated cliques becoming very numerous. In the next section, we present a new algorithm that avoids this duplicated search.

4 Extended clique maintenance algorithm

In this section, we describe improvements of the clique maintenance algorithm that were made to perform clique enumeration algorithm on a smaller subgraph G[fa(U,G)\W] than that of Ottosen and Vomlel (2010a). The clique enumeration algorithm is exponential with |V|. Therefore, this reduction of the search range is important. The proposed algorithm is described in Algorithm 3, where W is the set of vertices that have been eliminated.

The following example explains the algorithm:

Example 3. Consider again the middle-right graph in Figure 3. In our algorithm, the search range is limited to fa(U,G)\W = {B,C,D,E}. In this example, we only add cliques $\mathcal{NC}$(G,G') and remove cliques $\mathcal{RC}$(G,G').

Xiang and Lee (2006) describes a set of vertices called a crux which is central to their method for learning network structure. The algorithm described above may also be used to efficiently calculate the crux.

The correctness of this algorithm can be derived from the result below. Lemma 1 proves that our algorithm adds all new cliques $\mathcal{NC}$(G,G'). Lemma 2 proves that our algorithm removes all new cliques $\mathcal{RC}$(G,G'). Theorem 2 proves that our algorithm correctly updates the cliques for new graph.

Lemma 1. *Let G, G', F, and U be given as presented in Theorem 1. W are the vertices that have been eliminated. If C∈$\mathcal{NC}$(G,G'), then C⊆ fa(U,G')\W.*

Proof. Let C be a new clique, it must include at least a new edge, i.e. C includes a vertex v ∈U. Because C is complete, all vertices w∈C\U must be connected to the vertex v, i.e. C⊆ fa(U,G').

Presuming that C∈ $\mathcal{C}$(G'), which intersects with W, it must include a new edge, all vertices u∈W have been made simplicial by vertex elimination, which is a contradiction. Therefore, a new clique C⊆fa(U,G')\W. □

Lemma 2. *Let G, G', F, U, and W be given as in Lemma 1. If C∈ $\mathcal{RC}$(G,G'), then C intersects with U and disjoints with W.*

Proof. Let $C \in \mathcal{RC}(G, G')$. Assuming that C ∩ U = ∅, then for each v ∈ nb(C,G), C ⊈ nb(v,G')(otherwise C can not be a clique in G). Therefore, C is still a clique in G', which is a contradiction.

Let C∈ $\mathcal{RC}$(G,G'). Under assumption C∩W≠ ∅, for each w ∈C∩W, no v ∈nb(C,G') exists such that nb(v,G') includes w. Therefore, C is still a clique in G', which is a contradiction. Therefore, all remove clique C intersects with U and disjoint with W. □

Theorem 2. *Let G, G', F, U, and W be given as presented in Lemma 1. The $\mathcal{C}$(G') can be found by removing the cliques from $\mathcal{C}$(G) that intersects with U and disjoints with W, and then by adding cliques of G'[fa(U,G')\W] that intersect with U.*

Proof. (1) We first show that all cliques in $\mathcal{NC}$(G,G') are added. From lemma 1, the subgraph G'[fa(U,G')\W] includes all cliques in $\mathcal{NC}$(G,G'). Furthermore, any new clique C must intersect with U (otherwise it could not include a new edge). Therefore, we add all cliques in $\mathcal{NC}$(G,G').

(2) We show that all cliques in $\mathcal{RC}$(G,G') are removed. From lemma 2, if C ∈ $\mathcal{RC}$(G,G'), then C ∩ W = ∅ and C ∩ U ≠ ∅. Therefore, we remove all potential cliques in $\mathcal{RC}$(G,G').

(3) We consider that we can also remove a clique C ∈ $\mathcal{C}$(G') ∩ $\mathcal{RC}$(G). Because C intersects with U and disjoints with W, then C ⊆ fa(U,G')\W, and C will be added again when we add cliques from G'[fa(U,G')\W] that intersects with U. □

5 Search graph reduction

In this section, we exploit the following theorem to reduce the search tree:

Theorem 3. *(Dirac, 1961) A triangulated graph with at least two nodes has at least two simplicial nodes.*

Theorem 3 is applicable directly to depth-first search, as shown in algorithm 4.

Algorithm 4 Depth-first search with pruning.

```
1: Insert lines 1–8 of Algorithm 1
2: procedure ExpandNode(n,&best,&map)
3:     Let u = SelectOneVertex(n.R)
4:     for all v ∈ V(n.R)\u do
5:         Let m = Copy(n)
6:         EliminateVertex(m,v)
7:         EliminateSimplicial(m)
8:         Insert lines 14–23 of Algorithm 1
```

Table 1: Results for random Bayesian networks.

vertices	proposed (second)	Ottosen's (second)
30	5.62	12.08
35	20.77	33.82
40	46.39	97.3
45	85.37	187.29
50	220.08	418.68
55	1256.17	2158.18
60	36324	None

Recalling that a node represents an elimination order prefix, then for an arbitrary vertex u which is to be eliminated next, according to Theorem 3, we can prune the corresponding node (line 4 in Algorithm 4). This pruning based on the fact: For any elimination order prefix τ={ $v_1, v_2, \ldots, v_{i-1}$, t}, there exists another τ'={ $v_1, v_2, \ldots, v_{i-1}$, t'} (t'≠t) can engender the same triangulation. As a result, the search tree size can be reduced from |V|! to |V-1|!. As Ottosen and Vomlel (2010b) pointed out, the Branch and Bound with coalescing of nodes actually runs in $O(2^{|V|})$ time. Therefore, the actual runtime in this study is improved from $O(2^{|V|})$ to $O(2^{|V-1|})$.

6 Experiments

In this section, we present experimental results obtained from running our algorithm on random Bayesian networks (Ide and Cozman, 2002) and on benchmark graphs from the Bayesian network repository. For comparison, we also solve each triangulation problem using Ottosen's algorithm. The algorithms were implemented using Java. All experiments were conducted on a Xeon-5675 3.0 GHz processor (Intel Corp.) with 12 GB of RAM.

Table 2: Results for Bayesian networks from the repository.

BN	vertices	proposed (second)	Ottosen's (second)
Insurance	27	4.201	6.872
Water	32	21.696	65.682
Mildew	35	19.915	67.433
Alarm	37	0.006	0.012
Hailfinder	56	21.871	70.004
HEPAR2	70	0.054	0.115
WIN95PTS	76	158.259	299.181

All graphs that are described in Table 1 were generated using the BNGenerator system implemented by Ide and Cozman (2002). We generated 10 random graphs respectively for 30, 35, 40, 45, 50, 55 and 60 vertices. Each value in Table 1 shows the average time for the 10 graphs given the number of vertices, where "None" indicates that the algorithm obtained no solution in 24 hours. Additionally, we used seven well-known graphs in the Bayesian network repository. The results are presented in Table 2. The experiment results show that our algorithm is about 2 or 3 times faster than Ottosen et al.'s algorithm. The results also show that our algorithm can obtain the solution when |V| = 60 and the Ottosen et al.'s algorithm can not obtain it in 24 hours.

Figure 5 shows the log–log plot of running time of proposed algorithm and that of Ottosen's algorithm for all random and repository networks, on which the x-axis is the our running time for triangulation, and the y-axis is Ottosen's running time. The values above the line indicate that our proposed method is superior to Ottosen's method. From Figure 5, our proposed algorithm is shown to be constantly faster than Ottosen's approach for both random Bayesian networks and repository networks.

7 Conclusions

The existing state-of-the-art algorithms for optimal triangulation are depth-first search and best-first search. Ottosen and Vomlel (2010b) pointed out that depth-first search is better than best-first search and proposed a depth-first search algorithm of which the complexity is $O(\beta \cdot |V|!)$, where β is the per-node overhead

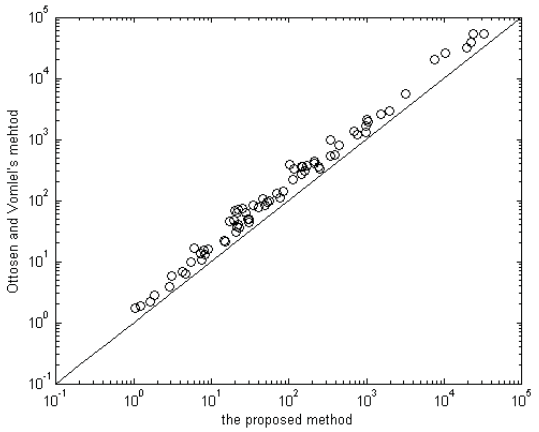

Figure 5: Comparison of the running times of Ottosen's method and the proposed method. X-axis shows results obtained using our method; the y-axis shows those obtained using Ottosen's method. Values above the line indicate that our method is faster.

computation. We developed a new algorithm for the optimal triangulation of a Bayesian network. The proposed algorithm improves the computational complexity of the Ottosen's algorithm in two ways.

For reduction of the computational cost β per node (section 4), we developed a new algorithm for maintaining the cliques of a dynamic graph. The new method is superior to Ottosen's method because it searches cliques on a smaller local graph than Ottosen's method does. This improves the computation cost from β to γ, where γ is strictly smaller than β.

For search tree shrinkage (section 5), we reduced the size of search tree from |V|! to (|V|-1)! using Dirac's theorem.

The complexity of our proposed triangulation algorithm is $O(\gamma \cdot (|V|-1)!)$. Some numerical experiments demonstrated that the proposed method improves the traditional method.

For the future work, we will investigate other implementations of search graph reduction, e.g. the way proposed in (Bodlaender et al., 2006) may improve our algorithm in running time.

References

H. L. Bodlaender, A. M.C.A. Koster, and F. V. D. Eijkhof. 2005. Preprocessing rules for triangulation of probabilistic networks. *Computational Intelligence*, 21(3):286–305.

H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. 2006. On exact algorithms for treewidth. In *Proceedings of the 14th conference on Annual European Symposium*, volume 14, pages 672–683.

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

A. Darwiche. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

G. A. Dirac. 1961. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universitat Hamburg*, 25:71–76.

P. A. Dow and R. E. Korf. 2007. Best-first search for treewidth. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, volume 2, pages 1146–1151. AAAI Press.

V. Gogate and R. Dechter. 2004. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 201–208, Arlington, Virginia, United States. AUAI Press.

J. S. Ide and F. G. Cozman. 2002. Random generation of bayesian networks. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, pages 366–375.

F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282.

T. J. Ottosen and J. Vomlel. 2010a. Honour thy neighbour—clique maintenance in dynamic graphs. In *Proceedings of the Fifth European Workshop on Probabilistic Graphical Models*, volume 2010-2. HIIT Publications.

T. J. Ottosen and J. Vomlel. 2010b. All roads lead to rome—new search methods for optimal triangulation. In *Proceedings of the Fifth European Workshop on Probabilistic Graphical Models*, volume 2010-2. HIIT Publications.

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

Y. Xiang and J. Lee. 2006. Learning decomposable markov networks in pseudo-independent domains with local evaluation. *Machine Learning*, 65:199–227.