Contents lists available at ScienceDirect
Theoretical Computer Science
journal homepage: www.elsevier.com/locate/tcs

Simple algorithms for minimal triangulation of a graph and backward selection of a decomposable Markov network

Mauro Mezzini *, Marina Moscarini
Department of Computer Science, Sapienza University of Rome, Italy[1]

ARTICLE INFO

Article history:
Received 30 December 2008
Received in revised form 7 October 2009
Accepted 8 October 2009
Communicated by G. Ausiello

Keywords:
Chordal graphs
Minimal triangulations
Markov networks
Learning decomposable models

ABSTRACT

In this paper we propose a simple algorithm called CliqueMinTriang for computing a minimal triangulation of a graph. If F is the set of edges that is added to G to make it a complete graph K_n then the asymptotic complexity of CliqueMinTriang is $O(|F|(\delta^2 + |F|))$ where δ is the degree of the subgraph of K_n induced by F. Therefore our algorithm performs well when G is a dense graph. We also show how to exploit the existing minimal triangulation techniques in conjunction with CliqueMinTriang to efficiently find a minimal triangulation of nondense graphs. Finally we show how the algorithm can be adapted to perform a backward stepwise selection of decomposable Markov networks; the resulting procedure has the same time complexity as that of existing similar algorithms.

1. Introduction

Computing a triangulation of a graph has several important applications in different areas such as nondense matrix computations [16], database management [1,19] and artificial intelligence [18,7,14]. A *triangulation* of a graph G is a chordal graph (i.e., a graph that contains no induced chordless cycle on four or more vertices) that has G as a subgraph (see [5] for a tutorial on chordal graphs).

For example large symmetric systems of equations arise in many engineering areas. Here the standard linear algebra method of the Gaussian elimination is employed to solve these systems. During the Gaussian elimination some cells of the matrix which are zero become non-zero. The number of cells that become non-zero heavily depends on the order in which the pivots are chosen in each step of the elimination. Some orderings insert much less non-zero entries than others thus saving space and reducing the computation effort needed for the Gaussian elimination. Given a symmetric matrix one can construct an undirected graph in which an edge ij with $i \neq j$ corresponds to a non-zero entry at the column i and row j of the matrix. Then one can simulate the Gaussian elimination on this graph by choosing at each step a vertex v and adding edges in the neighborhood of v until it becomes a clique before eliminating v from the graph. The added edges correspond to the entries of the matrix that become non-zero and this process is called Elimination Game. Thus the problem here is to find an ordering of the vertices of G such that the number of edges added during Elimination Game, is minimum. The graph produced by Elimination Game is a triangulation of the input graph.

In general, in all the above applications we are interested in solving the *minimum triangulation problem*, i.e. the problem of finding a triangulation H of a given graph G such that the difference between the number of edges of H and G is minimum. Unfortunately the minimum triangulation problem is NP-hard [20]. Therefore, one can try to find a *minimal triangulation* [2–4,9,10,16], that is, a minimal (w.r.t. set inclusion) set F of edges which added to a graph G makes it a chordal graph. This problem is solvable in polynomial time and has been widely studied in the past thirty years [9].

* Corresponding author. Tel.: +39 06 71036835.
E-mail address: mezzini@di.uniroma1.it (M. Mezzini).
[1] http://www.di.uniroma1.it/

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.10.004

Chordal graphs are also studied in the field of artificial intelligence and machine learning [14,8,18,7]. In this context one wants to estimate an n-dimensional discrete probability distribution from a finite set of given marginals in order to use a small amount of machine memory.

To this aim, one models joint probability distributions of n discrete random variables by *Markov networks* (also called *graphical models*). Such models use undirected graphs to capture conditional dependencies among subsets of the n random variables involved. Particular Markov networks, called *decomposable Markov networks* (DMNs), use chordal graphs. DMNs enjoy a number of desirable properties, one of which is that the approximate distribution has a simple 'product form' [15]. Therefore, given a joint probability distribution, one is interested in finding an approximation of it which is a decomposable Markov network. A possible approach, called *backward selection*, to solve this problem is the following: starting from the complete graph on n vertices (no assumption of independence among the random variables of the joint distribution is made) one recursively removes an edge from the graph while chordality is preserved.

The contribution of this paper is threefold. First, we present a simple algorithm for finding a minimal triangulation of a graph that performs well when the graph is dense (i.e, the difference between the edges of the complete graph and the edges of the input graph is small). Second we show how to use the above algorithm in conjunction with existing minimal triangulation techniques to efficiently find a minimal triangulation of a nondense graph. Third, we show that the algorithm can be easily adapted to get a backward selection procedure in order to find a DMN (of a given joint probability distribution) that minimizes the use of memory space and, at the same time, approximates the original distribution with a small error. Our algorithm has the same time complexity as previous similar algorithms [7,8] but it is simpler to understand and easier to implement.

The work is organized as follows. In Section 2 we give definitions and preliminaries. In Section 3 we describe our minimal triangulation algorithm, prove its correctness and discuss its implementation and its time complexity. In Section 4 we show how to use our algorithm in conjunction with existing minimal triangulation techniques to efficiently find a minimal triangulation of a nondense graph. In Section 5 we reports the data of experimental tests of our algorithm. In Section 6 we show how to use the algorithm to efficiently perform backward selection.

2. Definition and preliminaries

Let G be an undirected loopless simple graph. The *vertex set* and the *edge set* of G are denoted by $V(G)$ and $E(G)$, respectively; furthermore $n = |V(G)|$ and $m = |E(G)|$. A *clique* of G is a set of pairwise adjacent vertices. A graph is *complete* if its vertex set is a clique. The complete graph on n vertices is denoted by K_n. If F is a subset of $E(G)$, V_F denotes the set $\{x \in e : e \in F\}$, i.e. the set of vertices of G that are end points of an edge in F.

A graph H is a *subgraph* of G if $V(H)$ is a subset of $V(G)$ and $E(H)$ is a subset of $E(G)$. A graph G is a *supergraph* of H if H is a subgraph of G. Let U be a subset of $V(G)$ and F a subset of $E(G)$. By $G(U)$ and $G(F)$ we denote the subgraphs of G *induced* by U and F respectively; $G - U$ and $G - F$ denote the subgraphs $G(V(G) - U)$ and $(V(G), E(G) - F)$, respectively. Furthermore, if F is a subset of $E(K_n)$, $G + F$ denotes the graph $(V(G), E(G) \cup F)$.

The *neighborhood* of a vertex u in G is denoted by $N_G(u) = \{v : uv \in E(G)\}$. The *common neighborhood* in G of an edge uv, denoted by $CN_G(uv)$, is the set $N_G(u) \cap N_G(v)$. The *degree* of a graph, denoted by $\delta(G)$ is $\mathbf{max}\{|N_G(u)| : u \in V(G)\}$.

A *cycle* in a graph is a sequence $(v_1, \ldots, v_k)$ of at least four vertices, which are all distinct except the first and the last which do coincide, and such that $v_i v_{i+1}$ is an edge of G, for $1 \le i < k$. The *length* of a cycle is the number of its distinct vertices. A *chord* of a cycle is an edge joining two non-consecutive vertices of the cycle.

A graph is *chordal* or *triangulated* if every cycle of length four or more has a chord. A *triangulation* of a graph G is a chordal supergraph H of G with the same vertex set as G; a triangulation H of G is *minimal* if no subgraph of H is a triangulation of G.

Definition 1. Let G be a chordal graph. An edge uv is removable from G if it is not the unique chord of any 4-cycle of G.

Lemma 1 ([16]). *Let H be a triangulation of G. Then $E(H) - E(G)$ has at least one removable edge if and only if H is not minimal.*

Lemma 2 ([12]). *Let G be a chordal graph and uv an edge of G. Then uv is removable if and only if $CN_G(uv)$ either is empty or is a clique of G.*

3. A minimal triangulation algorithm

In this section we propose an algorithm that exploits Lemma 1 in order to compute a minimal triangulation of a graph G with n vertices starting from K_n and repeatedly deleting a removable edge not belonging to $E(G)$ until no more edge can be removed.

3.1. The algorithm

The input of the algorithm is the graph G and the set F of edges that added to G make it a complete graph. The output is a set $F' \subseteq F$ such that $G + F'$ is a minimal triangulation of G.

During the execution of the algorithm the set variable F' contains the edges in F not yet removed. By Lemma 2, an edge uv of F' is removable if and only if $CN_{G+F'}(uv)$ is a clique or is empty.

Observe that if $CN_{G+F'}(uv)$ is a clique or is empty then $CN_{G+F'}(uv) \cap V_F$ is a clique or is empty. On the other hand, owing to the fact that only edges in F are removed, if $CN_{G+F'}(uv)$ is not empty and is not a clique, then $CN_{G+F'}(uv) \cap V_F$ is not empty and is not a clique. Therefore, in order to check if an edge uv is removable, it is sufficient to test if $CN_{G+F'}(uv) \cap V_F$ is a clique or is empty. To this aim, the algorithm associates with each edge uv of F a variable T_{uv} that contains the edges in $F - F'$ (that is, the set of removed edges) that have both end points in $CN_{G+F'}(uv) \cap V_F$ and simply tests whether T_{uv} is empty (uv is removable) or not (uv is not removable).

When an edge uv is removed from F' it is necessary to add it in every T_{rs} such that $u, v \in CN_{G+F'}(rs) \cap V_F$. Furthermore, observe that if an edge uz has both end points in $CN_{G+F'}(vx) \cap V_F$ (and, hence, is in T_{vx}), after the removal of uv it has only one end point in $CN_{G+F'}(vx) \cap V_F$. Therefore, when uv is removed every edge uz having both end points in $CN_{G+F'}(vx) \cap V_F$ must be removed from T_{vx} and, analogously, every edge vz having both end points in $CN_{G+F'}(ux) \cap V_F$ must be removed from T_{ux}.

Finally, observe that if neither u nor v is in $CN_{G+F'}(xy) \cap V_F$ or $\{x, y\} \cap \{u, v\} = \emptyset$ then the set of edges in $F - F'$ that have both end points in $CN_{G+F'}(xy) \cap V_F$ (i.e. T_{xy}) does not change, or at least does not decrease, when uv is removed. The scheme of our algorithm is showed in Fig. 1. By the above discussion and by Lemmas 1 and 2 we can state the following

Theorem 1. *Let F be a subset of $E(K_n)$. The algorithm* CliqueMinTriang *correctly computes a set of edges $F' \subseteq F$ such that $(K_n - F) + F'$ is a minimal triangulation of $K_n - F$.*

```
Algorithm CliqueMinTriang
input: A graph $G$ and the edge set $F$ such that $G = K_n - F$
output: A set $F' \subseteq F$ such that $G + F'$ is a minimal triangulation of $G$

begin
 $F' := F$;
 for every $uv \in F'$ do $T_{uv} := \emptyset$;
 while there exists an edge $uv$ in $F'$ such that $T_{uv} = \emptyset$ (i.e. $uv$ is removable ) do
    begin
      for every $rs$ such that $u, v \in CN_{G+F'}(rs) \cap V_F$ do $T_{rs} := T_{rs} \cup \{uv\}$;
      for every $x \in CN_{G+F'}(uv) \cap V_F$ such that either $ux, vx \in F'$ do
         begin
           if $vx \in F'$ then for every $uz \in T_{vx}$ do $T_{vx} := T_{vx} - \{uz\}$;
           if $ux \in F'$ then for every $vz \in T_{ux}$ do $T_{ux} := T_{ux} - \{vz\}$;
         end
      $F' := F' - \{uv\}$;
    end
end
```

Fig. 1. The algorithm CliqueMinTriang

3.2. Implementation details and complexity

In the following let $f = |F|$ and let $\delta = \delta(K_n(F))$. We may store in a variable S_{uv} the elements of $CN_{G+F'}(uv) \cap V_F$ during all the steps of the algorithm. At the beginning, we set $S_{uv} = CN_{K_n}(uv) \cap V_F = V_F - \{u, v\}$ for each $uv \in F$. After deleting an edge uv we need to delete u from S_{vx} and v from S_{ux} for all $x \in S_{uv}$ and for all $ux \in F'$, $vx \in F'$.

Let us analyze the complexity of CliqueMinTriang. Computing all sets $S_{uv} = CN_{K_n}(uv) \cap V_F$ for all uv of F requires a global $O(|V_F|f)$ time. The **while** loop is executed at most f times. The first **for** loop statement finds all the sets S_{rs} that contain both end points of the edge uv and it adds uv to T_{rs}. All this requires $O(f)$ time .

The second **for** loop statement removes from every T_{vx} (resp. T_{ux}) such that $x \in S_{uv}$ and $vx \in F'$ (resp. $ux \in F'$) all the edges uz (resp. vz) such that $uz \in T_{vx}$ (resp. $vz \in T_{ux}$). All this requires $O(\delta^2)$ time.

Deleting the vertex u and v from S_{vx} and S_{ux} respectively, for all $x \in S_{uv}$ requires $O(\delta)$ time. Therefore we can state the following

Theorem 2. CliqueMinTriang *has $O(f(\delta^2 + f))$ time complexity.*

Although the complexity of CliqueMinTriang is greater than the complexity ($O(n^{2.376})$ [11]) of the fastest known minimal triangulation algorithm, CliqueMinTriang can be indeed very fast if the difference between the number of edges of the complete graph on $|V(G)|$ vertices and the number of edges of G is small, i.e., when G is dense.

In general we can see our algorithm as a dynamic procedure that, starting from the complete graph, deletes edges while preserving chordality. The complexity of our algorithm is comparable to that of existing procedures. In fact it requires $O(1)$ time to find an edge that can be removed from the graph while preserving the chordality and requires $O(n^2 + m)$ time to remove it. Therefore it works better than the algorithm proposed by Ibarra [12] that requires $O(nm)$ time to find a removable edge and $O(n)$ time to remove it. In fact, if we want to remove k edges then the overall complexity of our algorithm is $O(k(n^2 + m)) = O(kn^2)$ while the algorithm of Ibarra requires $O(knm) = O(kn^3)$.

Example 1. Consider the graph G with $E(G) = \{ab, bc, cd\}$. Then the set of edges needed to make the graph complete is $F = \{ac, bd, ad\}$ and $V_F = \{a, b, c, d\}$. The sets S_{uv} at the beginning of the while loop are reported in the following table.

uv	S_{uv}
ac	b, d
bd	a, c
ad	b, c

The algorithm chooses to remove first ac. Then it adds ac to T_{bd} since S_{bd} is the only set that contains both a and c. Next it removes c from the set S_{ad}. After this the algorithm stops the first iteration of the while loop, since all S_{cx} are empty. At the second iteration of the while loop the only edge that can be removed is the edge ad. No set S_{xy} contains the end points of the edge ad so the procedure eliminates a from S_{bd} and deletes ac from T_{bd}. After the elimination of ac, T_{bd} becomes empty. So the edge bd at the end of the second while loop iteration becomes again removable. At the third iteration of the while loop the only removable edge is bd. It will be removed and the algorithm terminates. ■

4. Minimal triangulation of nondense graphs

As we discussed above, CliqueMinTriang performs well when the difference between the number of edges of the complete graph and the number of edges of the graph to be triangulated is small but may have poor performance when this difference is big. To overcome this difficulty we propose to use an existing minimal triangulation algorithm, MinimalChordal [4], in conjunction with CliqueMinTriang. For ease of understanding, we report MinimalChordal in Fig. 2; we refer the reader to [4] for all the details.

Initially, MinimalChordal computes a triangulation of the input graph G, using the Elimination Game (EG) algorithm, which we recall below.

EG, given a graph G and an ordering $\alpha = (v_1, \dots, v_n)$ of its vertices, examines the vertices of G following α and recursively performs the following steps:

- let F_i be the set of edges not in G necessary to make $N_G(v_i)$ a clique.
- add F_i to G
- delete v_i from G

Let $G^+_\alpha = (V(G), E(G) \cup F)$, where $F = \cup_{i=1}^n F_i$, denotes the graph produced by EG with input G and α and $C_i = N_{G^+_\alpha}(v_i) \cap \{v_{i+1} \dots, v_n\}$. The edges of F are called *fill* edges.

After computing G^+_α, MinimalChordal analyzes the edges in F in reverse order (i.e., the edges in F_i are considered before the edges in $F_j, j = i-1, \dots, 1$) and removes from G^+_α every removable edge.

Let M_i be the graph obtained by the algorithm at the beginning of step *i*. The algorithm puts in *Candidate*(i) all the removable edges of F_i, that is, all the edges of F_i that are not the unique chord on any 4-cycle of M_i. Then it removes from M_i the set of edges *Candidate*$(i)-$*Keepfill*(i), where *Keepfill*(i) is the set of edges of a minimal triangulation of the graph obtained by removing all edges in *Candidate*(i) from the subgraph of M_i induced by $V($*Candidate*$(i))$. Such a minimal triangulation can be computed using any MinimalTriangulation algorithm. In particular in [4] the authors propose the use of Lex-M [16], whose time complexity is $O(nm)$, and show that with this choice the complexity of MinimalChordal is $O(f(m+f))$.

Note that since C_i is a clique of M_i and $V($*Candidate*$(i))$ is a subset of C_i then $V($*Candidate*$(i))$ is a clique. So we propose to compute the set *Keepfill*(i) by using CliqueMinTriang instead of Lex-M.

Now let $\Delta = \mathbf{max}_i\{\delta(M_i(Candidate(i)))\} \leq \mathbf{max}_i\{|C_i|\}$ and $f = |F|$; we have the following complexity result.

Theorem 3. *If in* MinimalChordal *the* MinimalTriangulation *algorithm is* CliqueMinTriang *then the complexity of* MinimalChordal *is* $O(f(n+\Delta^2+f))$.

Proof. At step i of MinimalChordal to check if the edge $uv \in F_i$ is removable requires $O(\,|CN_{M_i}(uv) \cap \{v_{i+1}, \dots, v_n\}|)$; furthermore CliqueMinTriang removes a single edge in $O(\,|Candidate(i)| + \delta(M_i(Candidate(i)))^2)$ time. Since each fill edge is examined at most once, and since $|Candidate(i)| \leq |F_i| \leq f$, $\delta(G(Candidate(i))) \leq \Delta$ and $|CN_{M_i}(uv) \cap \{v_{i+1}, \dots, v_n\}| \leq n$ we have that the overall complexity of the algorithm is $O(f(n+\Delta^2+f))$ □

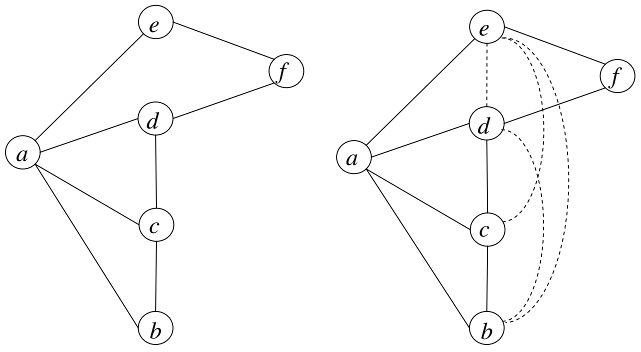
Therefore, in order to compute a minimal triangulation in MinimalChordal it is convenient to use CliqueMinTriang instead of Lex-M when Δ^2 is smaller than m. Moreover if $n_i = |V(Candidate(i))|$, $\delta_i = \delta(M_i(Candidate(i)))$, $f_i = |Candidate(i)|$ and $m_i = n_i(n_i-1)/2 - f_i$ we can devise a smarter algorithm that execute CliqueMinTriang whenever $f_i\delta_i^2 + f_i^2$ is less than $n_i m_i$ and Lex-M otherwise.

Example 2. Consider the graph of Fig. 3(a) and let $\alpha = (a, b, \dots, f)$. In Fig. 3(b) the chordal graph G^+_α is shown. We have that $F = F_1$. The set $Candidate(1) = \{bd, be, ce\}$ since the edge de is not removable (because af is not an edge of G^+_α). Then

```
Algorithm MinimalChordal
input: a graph $G$ and an ordering $\alpha$ of its vertices
output : A minimal triangulation $M$ of $G$ that is a subgraph of $G^+_\alpha$

begin
    $M_n := G^+_\alpha$;
    for $i := n$ downto 1 do begin
        $Candidate(i) := F_i$;
        for all edge $uv \in F_i$ do begin
            for each $x \in CN_{M_i}(uv) \cap \{v_{i+1}, \ldots, v_n\}$ do begin
                if $xv_i \notin E(M_i)$ then delete $uv$ from $Candidate(i)$;
            end
        end
        if $Candidate(i) \neq \emptyset$ then begin
            $Keepfill(i) :=$ MinimalTriangulation $(Candidate(i))$;
            $M_{i-1} := (M_i - Candidate(i)) + Keepfill(i)$;
        end
    end
end
```

Fig. 2. Algorithm MinimalChordal



(a) The original graph. (b) The graph after the execution of EG.

Fig. 3. Example 2.

$\{bd, be, ce\}$ is the input to CliqueMinTriang; in the following table the lists associated to candidate edges are reported.

uv	S_{uv}
be	c, d
bd	c, e
ce	b, d

Suppose that the algorithm chooses to remove ce. Then it adds ce to T_{bd} since S_{bd} contains both c and e (note that bd becomes now not removable). Next it removes c from the set S_{be}. After this the first iteration of the while loop terminates, since there is no set of the form S_{cx}. At the second iteration of the while loop the only edge that can be removed is be. No set contains the end points of be; so e is eliminated from S_{bd} and ce is eliminated from T_{bd}. Therefore, bd becomes again removable at the end of the second while loop iteration and it is removed at the third iteration. ■

5. Experimental results

By using C language we implemented both a version of MinimalChordal in which MCS-M [2] (we implemented MCS-M instead to LEX-M due to its simplicity) is used as MinimalTriangulation procedure and a version of MinimalChordal (in the following Mixed version) in which at each step either CliqueMinTriang or MCS-M is used as MinimalTriangulation procedure, depending on the estimated execution time. To this aim we modeled the execution cost of CliqueMinTriang and MCS-M by the following functions:

$$T_{CMT}(\delta_i, f_i) = \delta_i^2 f_i + f_i^2$$

and

$$T_{MCSM}(n_i) = n_i^3 + 3/2 n_i^2$$

Table 1
Results of execution of MinimalChordal,

Matrices	n	m	Initial fill	Final fill	MCS-M time	Mixed time
BCSSTK08	1074	5 943	156 778	71 319	18.342	12.236
BCSSTK09	1083	8 677	228 044	133 760	16.095	8.844
BCSSTK10	1086	10 492	104 545	94 199	0.686	0.639
BCSSTK11	1473	16 384	332 474	187 250	35.764	35.764
BCSSTK26	1922	14 207	111 918	46 525	3.173	3.000
BCSSTM07	420	3 416	29 166	12 287	0.674	0.674
CAN__838	838	4 586	86 175	75 922	1.469	0.939
CAN_1054	1054	5 571	68 935	23 727	7.098	5.881
CAN_1072	1072	5 686	71 736	28 023	7.155	6.222
DWT_1005	1005	3 808	95 837	45 030	10.692	5.984
DWT_1242	1242	4 592	83 925	30 781	6.995	3.829

Table 2
Detailed data for the first thirty vertices during the execution of MinimalChordal with input BCSSTK08 matrix.

$\|F_i\|$	n_i	f_i	δ_i	CMT time	MCSM time
301	302	301	301	0.016	0.125
319	320	319	319	0.016	0.140
321	307	306	306	0.015	0.126
680	342	680	340	0.312	0.172
375	376	375	375	0.016	0.187
377	378	377	377	0.015	0.187
379	380	379	379	0.016	0.203
777	395	764	382	0.469	0.219
388	369	368	368	0.016	0.187
776	370	735	368	0.391	0.218
400	373	372	372	0.015	0.188
4490	409	4412	400	4.516	0.280
394	384	383	383	0.015	0.204
388	389	388	388	0.016	0.218
786	403	786	394	0.500	0.234
782	398	782	391	0.485	0.234
393	387	386	386	0.016	0.218
1176	402	1150	392	0.531	0.250
780	399	760	390	0.031	0.235
392	360	359	359	0.016	0.187
792	404	757	389	0.484	0.250
1164	405	1104	380	0.968	0.251
396	357	356	356	0.016	0.187
395	363	362	362	0.015	0.188
396	353	352	352	0.000	0.187
790	387	705	358	0.048	0.202
397	356	355	355	0.016	0.172
412	413	412	412	0.031	0.235
414	396	395	395	0.015	0.235
835	370	702	363	0.344	0.188

We tested both implementations on some matrices from Matrix Market [6]. The results of the test are reported in Table 1, where the execution time of the two versions of MinimalChordal appears in MCS-M *Time* and Mixed *Time* column, respectively. As we can see, only in two cases the execution time is the same for both versions; in the remaining cases Mixed version performs better than the other one. This improvement of the execution time is due to the fact that, even if the input graph is sparse, the subgraph induced at the i-th step by $V(Candidate(i))$ is very dense and, as a consequence, CliqueMinTriang performs better than MCS-M. As an example of this fact, we reported in the first four columns of Table 2 the parameters $|F_i|, n_i, f_i, \delta_i$ for the first thirty vertices examined during the execution of MinimalChordal with input BCSSTK08 matrix and in the last two columns the execution time of CliqueMinTriang and MCS-M with input the subgraph induced at the i-th step by $V(Candidate(i))$, respectively.

6. A backward selection algorithm

Undirected graphs are commonly used to model joint probability distributions. These models, called *graphical models* or *Markov networks*, capture the conditional dependencies between the random variables involved in the joint distribution. A restricted class of graphical models of joint probability distributions are the *decomposable* [14] graphical models or decomposable Markov networks DMNs ([14,17]). These models possess a number of desirable properties that make them suitable for use in many practical applications. The problem we shall discuss is to find a DMN that models an approximation of a joint probability distribution.

In the following we explain the motivations of this problem and we show how CliqueMinTriang can be adapted in order to provide an heuristic to find a DMN that approximates a given joint probability distribution.

6.1. Graphical models

We are given a set X of random variables. Each variable X_i takes on values from a finite set called the *domain* of X_i. A tuple x on X is a combination of values of the variables in X. The set of all possible tuples on X, denoted by $dom(X)$, is given by the cartesian product of the domains associated to the variables in X. A *joint distribution* of X is a nonnegative function $p : dom(X) \to \Re$ such that $p(x) \leq 1$ for all x and $\sum_{x \in dom(X)} p(x) = 1$. We denote by $R = \{x \in dom(X) : p(x) > 0\}$ the *support* [15] of the distribution p. Given a nonempty subset Y of X the *restriction* of a tuple x to Y, denoted by x_Y, is the tuple obtained by discarding from x the values of variables in $X - Y$. A *marginal* distribution p_Y of p with respect to Y, is the distribution

$$p_Y(y) = \sum_x p(x)$$

where the sum is intended to be taken over all tuples x such that $x_Y = y$.

Note that storing the distribution p could require an exponential amount of memory space since the support of p can be as large as $dom(X)$. In order to save memory space one wants to store only a set of marginals, which could require far less memory space, and to use them to obtain an (accurate) approximation of the original distribution.

So the problem here is to find a set of marginals such that the memory space required in order to store them is not greater than a given constant and the difference between the original distribution and the approximated distribution provided by the marginals is small. The accuracy of the approximation is commonly measured using the *information divergence*, also known as the Kullback–Leibler (KL) *distance* or *divergence* [13].

In order solve this problem DMNs can be used. The benefit of using DMNs is that the approximation of the original probability distribution can be calculated analytically from marginal probabilities without resorting to the iterative proportional fitting procedures [15]. Furthermore, the KL divergence has a simple analytical expression [14] as we show in the following.

A DMN is a chordal graph $G = (V(G), E(G))$ where $V(G) = X$ and if A, B and C are sets of random variables and C separates A and B in G, then the set variables A and B, conditioned on C, are independent. It is well known that if $\mathcal{K} = \{K_1, K_2 \ldots, K_m\}$ is the set of maximal cliques of a chordal graph G and $\mathcal{T} = (\mathcal{K}, \mathcal{E}_{\mathcal{T}})$ is the *clique tree* [5] of G then for every pair of vertices K_i and K_j of $\mathcal{T}$ the set $K_i \cap K_j$ is contained in every K_l along the path on $\mathcal{T}$ that connects K_i and K_j. It is also known that given an edge K_iK_j of $\mathcal{T}$ then $K_i \cap K_j$ is a minimal separator of G. Denote by S_{ij} the set $K_i \cap K_j$ for all $K_iK_j \in \mathcal{E}_{\mathcal{T}}$. Then the approximation p_μ of a given distribution p modeled by the DMN G, is given by [14,15]

$$p_\mu(x) = \prod_{K \in \mathcal{K}} p_K(x_K) / \prod_{\substack{S_{ij} \\ K_iK_j \in \mathcal{E}_{\mathcal{T}}}} p_{S_{ij}}(x_{S_{ij}})$$

and the entropy of the distribution p_μ is given by [14]

$$H_\mu(X) = \sum_{K \in \mathcal{K}} H(K) - \sum_{\substack{S_{ij} \\ K_iK_j \in \mathcal{E}_{\mathcal{T}}}} H(S_{ij}) \tag{1}$$

where $H(A)$ is the entropy of the marginal distribution of the variables in A [14], that is

$$H(A) = -\sum_{x_A} p_A(x_A) \log p_A(x_A)$$

The KL divergence of p_μ from p is given by the difference between the entropy of p and the entropy of p_μ [14]. Since the entropy of p is independent of the model chosen as approximation, seeking a DMN that minimizes the KL divergence is equivalent to seeking a DMN whose entropy is minimum.

The amount of memory space gained by storing the marginal distributions instead of the original distribution, depends on the maximum of the cardinalities of cliques in $\mathcal{K}$, i.e. on the *treewidth* of G.

The problem of finding the DMN of a treewidth k, $k > 1$, that optimally fits the original distribution is known to be an NP-hard problem [18]. So heuristic search techniques are usually used [14,7,8]. Most of these procedures are based on either

(1) the *forward selection* approach in which one starts from the smallest model (i.e., the graph whose edge set is empty) and adds edges as long as the new model is still decomposable and the divergence of the approximation decreases, or

(2) *backward selection* approach, in which one starts from the largest model (i.e., the complete graph) and deletes edges while preserving the decomposability until the divergence of the approximation is less than a given quantity and the treewidth of the resulting model is sufficiently small.

Table 3
Numerical comparisons of existing heuristic with the algorithm BSA.

k	Algorithm	Maximal cliques	Entropy
3	HGA	{ABCE, ACDE}	3.23282
	EGA	{ABDE, ACDE}	3.22640
	BSA	{ABDE, ACDE}	3.22640
2	HGA	{ABE, ACE, CDE}	3.24333
	EGA	{ABE, BDE, CDE}	3.24115
	BSA	{ABE, BDE, CD}	3.24995
1	HGA	{AE, BE, CD, CE}	3.26413
	EGA	{AE, BE, CD, CE}	3.26413
	BSA	{AE, B, CD}	3.28629

6.2. *Description of the algorithm and experimental comparison with existing heuristics*

In this subsection we show that CliqueMinTriang can be easily adapted to obtain a greedy backward selection algorithm (BSA) which has the same time complexity as previous [8,7] best forward selection algorithms, but is simpler to understand and easier to implement. We use CliqueMinTriang with input the set of edges $F = E(K_n)$ of the complete graph on n vertices. Let F' be the subset of edges of F not yet removed and let $M = (K_n - F) + F'$. Among all removable edges of F' one chooses to remove an edge that minimizes the KL divergence of the new model from the old one; the edge removal is repeated until a DMN of a given treewidth is obtained. The KL divergence of the new model from the old one can be computed as follows. Let uv be a removable edge of F' and let $K_M(uv) = \{u, v\} \cup CN_M(uv)$. It is easy to see that $K_M(uv)$ is a maximal clique of M and $K_M(uv) - \{u, v\}$ is a minimal separator of $M - uv$. Therefore, by applying (1), the KL divergence of $M - uv$ from M is (see for example [7])

$$\mathcal{D}(uv) = H(K_M(uv) - \{u\}) + H(K_M(uv) - \{v\}) - H(K_M(uv) - \{u, v\}) - H(K_M(uv)) \qquad (2)$$

For every pair uv, the value of $\mathcal{D}(uv)$ in K_n can be computed at the beginning of the algorithm. When an edge rs is removed, the algorithm updates the value of $\mathcal{D}(rx)$ and $\mathcal{D}(sx)$ only for those edges rx and sx of F' such that the set S_{rx} (S_{sx}, respectively) is modified and sx (rx, respectively) is still removable after the modification. This can be done without increasing the complexity of the whole procedure. Note that the content of the variable S_{uv} in the algorithm CliqueMinTriang is exactly $K_M(uv) - \{u, v\}$. Therefore, the overall complexity of removing a single edge is $O(n^2)$, so that the global complexity is $O(hn^2)$ where h is the number of edges removed by the algorithm.

In order to compare BSA with the forward selection algorithm proposed in [7,8] called EdgewiseGreedyAlgorithm (EGA) and the heuristic proposed in [14], called HyperedgeGreedy-Algorithm (HGA), we executed the algorithm BSA on the example appearing in [14] and also used in [8]. The results of the three algorithms are reported in Table 3; in the column Entropy is reported the entropy value of the DMN obtained in output for each algorithm.

The steps of the algorithm are as follows. In the first step all the edges are removable. So the edge that minimizes the entropy gives the optimal DMN of treewidth 3. At steps two, tree and four the edges AC, CE and AD, respectively, are removed giving the DMN of treewidth 2. Finally at steps five six and seven the edges BD, DE and AB, respectively, are removed giving the DMN of treewidth 1.

Acknowledgment

The authors wish to thank Franco Malvestuto for his useful comments.

References

[1] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, J. ACM 30 (3) (1983) 479–513.
[2] A. Berry, J.R.S. Blair, P. Heggernes, B.W. Peyton, Maximum cardinality search for computing minimal triangulations of graphs, Algorithmica 39 (4) (2004) 287–298.
[3] A. Berry, J.P. Bordat, P. Heggernes, G. Simonet, Y. Villanger, A wide-range algorithm for minimal triangulation from an arbitrary ordering, J. Algorithms 58 (1) (2006) 33–66.
[4] J.R.S. Blair, P. Heggernes, J.A. Telle, A practical algorithm for making filled graphs minimal, Theoret. Comput. Sci. 250 (1-2) (2001) 125–141.
[5] J.R.S. Blair, B.W. Peyton, An introduction to chordal graphs and clique trees, in: J.A. George, J.R. Gilbert, J.W.H. Liu (Eds.), Sparse Matrix Computations: Graph Theory Issues and Algorithms, in: IMA Volumes in Mathematics and its Applications, 56, Springer Verlag, 1993, pp. 1–30.
[6] R. Boisvert, R. Pozo, K. Remington, B. Miller, R. Lipman, NIST MatrixMarket. http://math.nist.gov/MatrixMarket/index.html.
[7] A. Deshpande, M.N. Garofalakis, M.I. Jordan, Efficient stepwise selection in decomposable models, In: UAI, 2001, pp. 128–135.
[8] G. Ding, R.F. Lax, J. Chen, P.P. Chen, B.D. Marx, Comparison of greedy strategies for learning markov networks of treewidth k, in: H.R. Arabnia, M. Dehmer, F. Emmert-Streib, M.Q. Yang (Eds.), Proceedings of the 2007 International Conference on Machine Learning, CSREA Press, Las Vegas Nevada, USA, 2007, p. 294.
[9] P. Heggernes, Minimal triangulations of graphs: A survey, Discrete Math. 306 (3) (2006) 297–317.
[10] P. Heggernes, B. Peyton, Fast computation of minimal fill inside a given elimination ordering, SIAM J. Matrix Anal. Appl. 2008, (in press).
[11] P. Heggernes, J. A. Telle, Y. Villanger, Computing minimal triangulations in time $o(n^{\alpha} \log n) = o(n^{2.376})$, SIAM J. Discret. Math. 19 (4) (2005) 900–913.
[12] L. Ibarra, Fully dynamic algorithms for chordal graphs and split graphs, ACM Trans. Algorithms 4 (4) (2008) 1–20.
[13] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Statist 22 (1) (1951) 79–86.

[14] F. Malvestuto, Approximating discrete probability distributions with decomposable models, IEEE Trans. Syst. Man Cybern. 21 (5) (1991) 1287–1294.
[15] F.M. Malvestuto, Existence of extensions and product extensions for discrete probability distributions, Discrete Math. 69 (1) (1988) 61–77.
[16] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (2) (1976) 266–283.
[17] N. Srebro, Maximum likelihood markov networks: An algorithmic approach, MSc Thesis, Massachusetts Institute of Technology. http://ttic.uchicago.edu/~nati/Publications/Mthesis.pdf2000.
[18] N. Srebro, Maximum likelihood bounded tree-width markov networks, in: UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, pp. 504–511, 2001.
[19] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (3) (1984) 566–579.
[20] M. Yannakakis, Computing the minimum fill-in is np-complete, SIAM. J. Algebr. Discrete Methods 2 (1) (1981) 77–79.