Mach Learn (2011) 84:249–289
DOI 10.1007/s10994-010-5233-4

Creating non-minimal triangulations for use in inference in mixed stochastic/deterministic graphical models

Chris D. Bartels · Jeff A. Bilmes

Received: 1 November 2008 / Revised: 21 April 2010 / Accepted: 14 September 2010 /
Published online: 8 February 2011

Abstract We demonstrate that certain large-clique graph triangulations can be useful for reducing computational requirements when making queries on mixed stochastic/deterministic graphical models. This is counter to the conventional wisdom that triangulations that minimize clique size are always most desirable for use in computing queries on graphical models. Many of these large-clique triangulations are non-minimal and are thus unattainable via the popular elimination algorithm. We introduce *ancestral pairs* as the basis for novel triangulation heuristics and prove that no more than the addition of edges between ancestral pairs needs to be considered when searching for state space optimal triangulations in such graphs. Empirical results on random and real world graphs are given. We also present an algorithm and correctness proof for determining if a triangulation can be obtained via elimination, and we show that the decision problem associated with finding optimal state space triangulations in this mixed setting is NP-complete.

Keywords Triangulation · Inference · Graphical models · Graph theory

1 Introduction

Probabilistic graphical models are a powerful tool for modeling factorized multivariate distributions. Factorizations are advantageous because they can constrain the parameters needed to represent these distributions. When factorization also leads to conditional independence, the time and memory needed to perform probabilistic computations can drop exponentially.

Graphical models often contain variables that have deterministic relationships with other variables in the graph. In particular, Bayesian networks can include variables whose value

Editor: Lise Getoor.

C.D. Bartels (✉) · J.A. Bilmes
Department of Electrical Engineering, University of Washington, Seattle, WA 98195, USA
e-mail: bartels@ee.washington.edu

J.A. Bilmes
e-mail: bilmes@ee.washington.edu

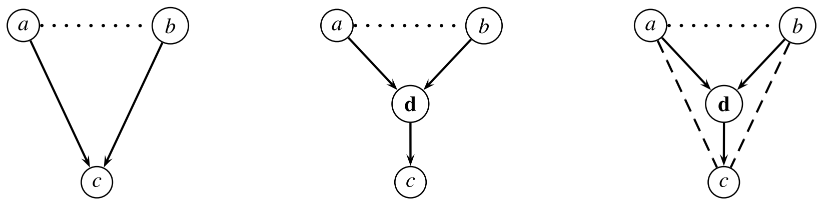
is a function of its parents. Such variables can be used to provide hard or soft constraints (Dechter 2003), or they can be used to factor a dense probability table into a product of smaller tables. Even when the graph designer does not specify any deterministic variables, methods have been developed to discover these factorizations (Vomlel 2002). Section 7.3 will describe how deterministic variables are used in a real-world automatic speech recognition system. In that example, the deterministic variables are used to constrain the allowable sequences of states in a dynamic Bayesian network.

The utility of deterministic variables has prompted a body of research on efficient calculations in mixed deterministic/stochastic graphs (Jensen and Andersen 1990; Dechter and Larkin 2001; Larkin and Dechter 2003; Bacchus et al. 2003a, 2003b; Dechter and Mateescu 2004). Computing probabilistic quantities can be done using a junction tree (Lauritzen and Spiegelhalter 1988; Pearl 1988), as a search procedure (Bacchus et al. 2003a, 2003b; Dechter and Mateescu 2004), or using a hybrid scheme (Dechter 2003). Search methods have been growing in popularity as they appear to be more effective at dealing with determinism and performing time/memory trade-offs (Bacchus et al. 2003a, 2003b; Sang et al. 2005; Chavira and Darwiche 2008). Junction trees can have advantages over search when a breadth first search is more desirable than a depth first search. An example of this is when dealing with sequential data in the context of hidden Markov models and other dynamic Bayesian networks (DBNs). The work we present here applies to any inference method that requires a junction tree. Of particular interest are hybrid schemes where a junction tree is formed, but rather than standard summations over tables, search methods are used for projecting clique potentials onto the separators. This scheme was given by Dechter for solving constraint satisfaction problems (Dechter 2003). Hybrid inference gives the advantages of junction trees but still allows most of the techniques for dealing with determinism that are available with search methods. Inference algorithms will be addressed again in Sect. 2.

As mentioned, we will be working with junction trees, and a junction tree can be formed if and only if the underlying graph is *triangulated*. If the graph is not triangulated, we must add edges to it until it becomes so. The choice of triangulation does not change the results of any probabilistic calculations, but it can make an exponential difference in the time and memory that is needed. Unfortunately, finding an optimal triangulation is NP-hard (Arnborg et al. 1987; Wen 1991) so heuristic search methods must be used. All exact inference methods in one way or another implicitly define at least one graph triangulation (Jensen and Jensen 1994). In the case of search methods, choosing the order the variables are instantiated together with a caching scheme gives an implied triangulation. In other cases, a search tree is formed by first defining a junction tree (Dechter and Mateescu 2004).

A common measure of triangulation quality is the size of the clique potentials formed in junction tree message passing, usually called the state space or weight (Lauritzen and Spiegelhalter 1988; Kjaerulff 1990; Wen 1991). Specifically, the **state space** or **cardinality** of a vertex v, notated $|v|$, is the number of distinct values it may hold. The state space of a *clique*, C, holding vertices $v_1, v_2, \dots, v_k$ is defined as $S(C) = \prod_{i=1}^{k} |v_i|$. Lastly, the state space of a graph, G, with maximal cliques $C_1, C_2, \dots, C_k$ is defined as $S(G) = \sum_{i=1}^{k} S(C_i)$.

State space is a good indicator of triangulation performance when dealing with standard junction tree inference on positive distributions. Distributions containing determinism are not positive, though. When determinism is present in a Bayesian network, a very basic optimization for inference comes from the fact that the value of a deterministic variable can be uniquely determined given its parents. To exploit this, one only needs to evaluate deterministic variables using their associated functions whenever possible rather than iterating over them as if they were stochastic. For a given parent combination it contributes only a constant time factor to evaluate the function, and performing this calculation does not require zero



(a) Non-factored. The graph has a state space of $|a||b||c|$.

(b) Factored. When **d** is deterministic, this graph has a state space of $|a||b| + |d||c|$.

(c) Triangulation of (b) that removes effect of factorization. When **d** is deterministic, this graph has a state space of $|a||b||c|$.

Fig. 1 Graph that demonstrates the use of a deterministic variable, **d**, to factorize a distribution. *Solid arrows* are original graph edges, *dotted lines* are moralization edges, and dashed lines are fill-in edges

compression or any form of constraint propagation. In a moralized Bayesian network, every variable will be in at least one maximal clique with its parents. A variable may also be a member of other maximal cliques because of its children and other non-parent neighbors. If a deterministic variable lives in a maximal clique that is missing one or more of its parents, its value cannot be determined uniquely and it does need to be treated just as if it were stochastic. We will now modify our definition of clique state space to reflect this:

Definition 1 The state space of a *clique*, C, with vertices $v_1, v_2, \dots, v_k$, and the set $\mathscr{D} = \{v | v \text{ is deterministic and parents of } v \in C\}$ is: $S(C) = \prod_{v \in C \setminus \mathscr{D}} |v|$.

This measure is still not fully adequate to describe the utility of a triangulation in every situation. The various methods mentioned for computing probabilities have different performances and a wide variety of time-space trade offs. Costs of queries on distributions with determinism are difficult to gauge without actually performing the computation. With this being said, this altered version of state space can give a measure of the upper bound of the computational requirements. The tightness of this bound depends heavily on the particular graph and will vary with different inference algorithms, but as will be seen we obtain significant wall-clock speedups under this assumption using a modern probabilistic inference engine.

When the altered state space is indicative of inference performance it presents problems for conventional triangulation techniques. Consider the graph in Fig. 1(b). This graph is triangulated and contains two maximal cliques, $\{a, b, d\}$ and $\{d, c\}$. The clique $\{a, b, d\}$ contains both deterministic variable d and its parents, so d does not contribute to the state space. In the clique containing $\{d, c\}$, the parents of d are not present so its value cannot be calculated and it does contribute. This gives a total state space of $|a||b| + |d||c|$. The graph in Fig. 1(c) is an alternative triangulation of Fig. 1(b). This graph has a single, large maximal clique with a state space of $|a||b||c|$. If d has a small cardinality relative to a and b the state space of 1(b) will be smaller than that of 1(c), but if d is large enough 1(c) is preferred. We show in Sect. 3 that on graphs with deterministic variables, triangulations with large cliques can use unboundedly less computational resources than triangulations that minimize clique size. In fact, when considering deterministic variables the state space optimal triangulation will often be completely outside of the search space of conventional triangulation techniques, including the popular elimination algorithm.

In the example from Fig. 1 the decision between 1(b) and 1(c) is straightforward, but if this is a part of a larger graph it is not so simple. It will be demonstrated in Sect. 6 that the optimal choice cannot be made locally and the entire fill-in must be considered to choose the correct triangulation. For the same reason, any method which uses graph transformations to remove the deterministic variables (such as in Fig. 1(a)) would need to consider the possible transformations in tandem with the possible triangulations of each transformed graph. Section 6.1 demonstrates how the search can be constrained by introducing *ancestral pairs*. Ancestral pairs are the basis for novel triangulation heuristics and it is proven that no more than the addition of edges between ancestral pairs needs to be considered when searching for state space optimal triangulations in such graphs. Empirical results on random and real world graphs are given in Sect. 7.

The paper also presents background material on inference, graph theory, and elimination in Sect. 2. Section 4 presents an algorithm and correctness proof for determining if a triangulation can be obtained via elimination, and Sect. 5 proves that the decision problem associated with finding optimal state space triangulations in this mixed setting is NP-complete. Section 8 describes a number of other methods for mixed stochastic/deterministic graph triangulation that could be explored in future research, and Sect. 9 extends the results to undirected graphs.

Some of the work presented herein was given by the same authors in Bartels and Bilmes (2006) and a related technical report (Bartels and Bilmes 2004). This article, however, presents several new results. A new set of results on randomly generated DBNs is presented in Tables 3 and 4. Section 8 on alternative methods for mixed stochastic/deterministic graph triangulation methods and Sect. 9 extending the previous work to undirected graphs are both new. Theorems 9 and 12 were not previously presented, and Theorems 7, 11 and 13 appear in the technical report but not in the conference publication. Section A.2 adds a description of the complexity of Algorithm 1. Expanded and more detailed descriptions, proofs, and additional diagrams are given throughout. Much of the paper was also presented in the thesis of Bartels (2008).

2 Background

Before presenting the results of this paper, we provide needed background on probabilistic inference. As mentioned, computing probabilistic quantities can be done using a search procedure (Bacchus et al. 2003a, 2003b; Dechter and Mateescu 2004), a junction tree (Lauritzen and Spiegelhalter 1988; Pearl 1988), or using a hybrid scheme (Dechter 2003). Search methods traditionally work by performing a backtracking search through the possible instantiations of the variables. This can be seen as an instance of the DPLL algorithm for satisfiability (Davis et al. 1962; Bacchus et al. 2003b). This concept was brought to graphical model inference in various forms in Pearl (1988), Dechter (1990), Darwiche (2001) and in particular as the Value Elimination algorithm in Bacchus et al. (2003b). Search methods were studied further as an exploration of an AND/OR solution space in Dechter and Mateescu (2004), Mateescu (2007) and as weighted model counting in Sang et al. (2005), Chavira and Darwiche (2008). These algorithms build caches of partial computations to exploit conditional independence. Methods also exist to calculate marginals for the entire distribution for the order of magnitude cost of a single query (Sang et al. 2005; Filali and Bilmes 2007). One of the primary advantages of search methods is that they are effective at dealing with determinism. First, since they work by instantiating all of the variables in the distribution the values of deterministic variables are always known and one can immediately evaluate constraints. In some cases this reveals context specific (local) structure that is not evident from the graph topology. Second, constraint propagation and "nogood" learning (Dechter 1990) can be used to reduce the size of the search space. Last, using depth first search allows such algorithms to evaluate variables in different orders when exploring different branches of the search tree.

The junction tree algorithm can be viewed as a generalization of the elimination algorithm developed much earlier for satisfiability and discrete optimization problems (Davis and Putnam 1960; Bertele and Brioschi 1972). Elimination was extended directly to probabilistic calculations in Dechter (1996). In elimination one performs a summation over each of the distribution's variables in turn. Each of these summations removes a variable from the distribution and creates an intermediate table that is used in the next summation. This procedure can be seen as a breadth first search over the solution space (Mateescu 2007). Message passing in a junction tree is also a series of summations. The nodes and separators of a junction tree provide an elegant and efficient data structure to cache the intermediate tables in a way that allows probabilistic quantities for the entire distribution to be calculated in the order of magnitude time/memory cost of an elimination calculation for a single variable. The primary criticism of junction trees is the memory requirements, but when one desires to query many or all the variables in the graph the memory in the junction tree is needed and any search algorithm will have similar storage requirements.

Many methods blur the line between the two types of inference. Cutset conditioning (Dechter 1990) and recursive conditioning (Darwiche 2001; Allen and Darwiche 2003) work by instantiating some variables and summing over others. Methods such as zero compression give structured representations to junction tree tables (Jensen and Andersen 1990) and methods such as Dechter and Larkin (2001), Larkin and Dechter (2003) can take advantage of determinism in this setting. There are also ways of exploiting context specific structure in junction trees and variable elimination (Boutilier et al. 1996; Chavira and Darwiche 2007). Alternatively, in Mateescu (2007) it was demonstrated how inference via breadth first search can be performed over AND/OR search spaces.

The form of hybrid inference most pertinent to the work presented here is based on the method described in Dechter (2003). In junction tree inference, the primary computation is in the projection of a clique potential onto a separator. This is the computation associated with a "message" and is a summation over all of the variables that are in the clique potential but not the separator. This is traditionally an iteration over all possible variable value combinations in the potential. When dealing with a positive distribution this can be implemented efficiently using a set of nested for loops. In hybrid inference, this summation is performed using a search procedure to take advantage of any sparsity in the potential. The search adds significant complexity, but can also yield significant savings when the distribution contains many entries with zero probability.

The methods given in this paper do not rely on using a search algorithm for projecting cliques onto separators. We only rely on making the following, simple optimization. Whenever a clique potential contains a deterministic variable and its parents, the deterministic variable's value is always computed directly. This can be done in a nested for loop implementation by simply replacing the for loop that iterates over the deterministic variable by an evaluation of the variable's associated function. Since there is no iteration over the values of the deterministic variable, its computational cost is a constant. Since a deterministic variable takes on the value of its function with probability 1 (given its parents), the probabilities associated with the deterministic variable's values do not need to be stored in a table and do not add any memory requirements to the clique potential.

This paper only discusses exact inference and does not touch upon approximate methods. In many cases, with a good triangulation we can run exact inference whereas many approximate inference algorithms have no guarantees as to their quality. Research in Boolean satisfiability (SAT) solvers continues to make progress and modern solvers are able to find exact solutions to problems that were not possible in the past (Zhang and Malik 2002; Berre and Simon 2005). Similarly, we believe progress can be made with continued work in exact probabilistic inference. Furthermore, many of the results here are graph-theoretic and apply to general graphs or for graphs with vertices marked with additional properties.

2.1 Graph theory and elimination

Before moving on to the main content of the paper, we define much of the notation, terminology, and known theorems required in later sections. We also discuss the elimination algorithm and relate it to our work herein.

A **graph** $G = (V, E)$ is defined as a set of vertices, V, and edges E. An edge is an unordered pair of vertices implying that $E \subseteq V \times V$. A **path** in a graph G is an ordered set of vertices such that each consecutive pair of vertices in the path is connected by an edge. A **cycle** is a path with two additional properties. These properties are that each vertex only appears once in the cycle and that there is an edge that connects the first and last vertices of the cycle. A **complete set** of vertices has an edge between every pair of vertices in the set. A graph $G_1 = (V_1, E_1)$ is a **subgraph** of $G_2 = (V_2, E_2)$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. A graph $G_1 = (V_1, E_1)$ is a **spanning subgraph** of $G_2 = (V_2, E_2)$ if $V_1 = V_2$ and $E_1 \subseteq E_2$.

A **chord** is an edge connecting two non-consecutive vertices in a cycle. A graph is **triangulated** if it contains no chordless cycles of length greater than three. A **triangulation** of a graph $G = (V, E)$ is a (possibly empty) set of edges $F \subseteq V \times V$ such that $E \cap F = \emptyset$ and the graph $T(G) = (V, E \cup F)$ is triangulated. The edges in F are called **fill-in edges**. The term triangulation will also be used to mean the graph $T(G)$. Given a graph $G = (V, E)$, the **neighbors** of $v \in V$ are defined as $NE_G(v) = \{w \in V | (v, w) \in E\}$. A triangulation F of graph $G = (V, E)$ is **minimal** if $G' = (V, E \cup F_0)$ is not triangulated for any $F_0 \subset F$. An edge e is a **non-minimal edge** in triangulation $T_1(G) = (V, E \cup F)$ if $T_2(G) = (V, E \cup F \setminus \{e\})$ is also triangulated. A **clique** is a set of vertices for which every vertex in the set is connected to every other vertex in the set. A **maximal clique** is a clique that is not a subset of some larger clique. The **treewidth** of a triangulated graph (Arnborg et al. 1987) is the size of its largest maximal clique minus 1. The treewidth of an arbitrary graph is the smallest treewidth of all possible triangulations. A vertex is **simplicial** in the graph G if $NE_G(v)$ form a complete set. Triangulated graphs with more than one node have at least two simplicial vertices (Golumbic 1980).

Vertex elimination (Parter 1961; Rose 1970) is an algorithm that can be used to triangulate graphs (Kjaerulff 1990; Larranaga et al. 1997; Meila and Jordan 1997; Dechter 1998). An **elimination order** is a bijection $\alpha : V \leftrightarrow \{1, 2, \ldots, |V|\}$. We use $\alpha(i)$ to denote the vertex indexed by the integer i in ordering α, and $\alpha^{-1}(v)$ to denote the integer position of node v in the ordering α. The **deficiency** (Rose 1970) of a vertex v in G is: $D_G(v) = (\{u, w\} | \{v, u\} \in E, \{v, w\} \in E, \{u, w\} \notin E\})$. That is, the deficiency of a vertex v is the set of edges that one would need to add in order to make the neighbors of v a complete set. Given $G = (V, E)$, the v**-elimination** graph G_v is defined by adding the edges $D_G(v)$ to G and then deleting v and its incident edges from G. Creating G_v from G is known as eliminating the vertex v. The **elimination graph**, denoted as $\xi_\alpha(G)$, is the original graph G with the addition of any edges added at each step in the elimination process according to α (see Rose 1970, $MTE(G; \alpha)$). A **perfect ordering** is an ordering for which elimination adds

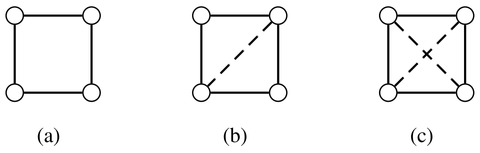


Fig. 2 Triangulation (**b**) is an elimination based triangulation of (**a**). Triangulation (**c**) of (**a**) is impossible with elimination. *Solid lines* are original graph edges and *dashed lines* are fill-in edges (Ohtsuki et al. 1976)

no edges, i.e., α is perfect if $\xi_\alpha(G) = G$. A perfect order need not exist for a general graph. The following theorems are important when working with elimination:

Theorem 1 *All elimination graphs are triangulated.* (Rose 1970)

Theorem 2 *A graph is triangulated if and only if it has a perfect elimination ordering* (Rose 1970, *Theorem* 1).

Although all elimination graphs are triangulated and all triangulated graphs have perfect elimination orders, some triangulations of a graph cannot be generated via *any* elimination order (Ohtsuki et al. 1976 and Fig. 2). Elimination can, however, create any minimal triangulation (Ohtsuki et al. 1976):

Theorem 3 *If $T(G) = (V, E \cup F)$ is a minimal triangulation of $G = (V, E)$, then there exists an elimination order α such that $\xi_\alpha(G) = T(G)$* (Ohtsuki et al. 1976, *Theorem* 1).

Most heuristics that do not involve elimination (such as Parra and Scheffler 1995; Bodlaender et al. 2001) will choose minimal triangulations over non-minimal triangulations.

In this section we have defined elimination in the graph theoretical sense. The term elimination is also used to describe a class of algorithms where variables are removed from a large expression one by one, typically by summing or maximizing over each variable in turn. An early elimination algorithm can be found in Davis and Putnam (1960), where variables are eliminated from Boolean formulas to determine if the expression is satisfiable. The work in Parter (1961) presented graph theoretical elimination as a way to represent Gaussian elimination on a sparse symmetric matrix, and he demonstrated that trees have perfect elimination orderings. Rose (1970) extended this result showing that a graph is triangulated if and only if it has a perfect elimination ordering. What is typically known as the elimination algorithm was presented by Bertele and Brioschi (1972)—this book presents the elimination algorithm as a general dynamic programming approach for discrete optimization problems and illustrates the approach using graph theoretic elimination. In Lauritzen and Spiegelhalter (1988), elimination is suggested as a fast way to triangulate a graph for the sake of forming a junction tree for inference. Dechter (1996, 1998), extended the methods of Bertele and Brioschi to a wide variety of probabilistic calculations.

Our interest in triangulation is for its use in one of the many exact probabilistic calculations that can be performed on a graphical model. Examples of these calculations are: 1) finding any configuration over the set of variables that has a non-zero probability (similar to constraint satisfaction (Dechter 2003) or the SAT problem (Cook 1971); 2) summing the scores of all variable assignments that do not get zero probability, useful for computing the p(evidence) (similar to weighted SAT (Sang et al. 2005)); 3) finding and storing all variable assignments that do not get zero probability, useful for various kinds of parameter learning (such as EM or gradient based), calculating posterior probabilities, and finding the N most probable variable assignments in the graph (Nilsson 1998); 4) finding the most probable assignment of all variables given the evidence (Viterbi 1967; Bertele and Brioschi 1972); 5) finding the most probable assignment of some of the variables given the evidence (which can be more expensive than the previous queries Park 2002). For each of the above queries, there are a variety of methods, each a compromise within a time-space trade off. For example, it is possible to compute the sum of all scores in time exponential in $|V|$ (the number of node variables) and linear memory in the number of nodes, or alternatively time and space exponential in w (the treewidth).

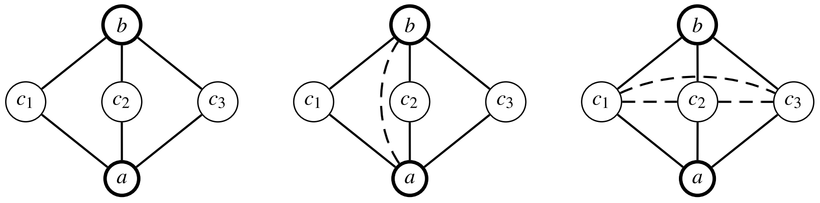
3 Minimal versus non-minimal triangulations

Given a background on triangulation and elimination, we can continue the discussion of when and why one would want to move beyond conventional triangulation techniques. It is a common belief that triangulations that minimize clique size are always desirable for use in computing queries on a graphical model. The reason is that it can be shown that many of the metrics of performance are upper bounded in some exponential function on w, the inherent treewidth of the graph (Darwiche 2001; Dechter and Mateescu 2004; Dechter 1998). A triangulation with minimum treewidth is *not* necessarily optimal for probabilistic queries. An instance of this is when it may be more desirable to cluster many small cardinality variables together in a clique to avoid increasing the sizes of cliques that contain large cardinality variables (see Fig. 3). If the cardinalities of the variables differ dramatically, upper bounds derived from the treewidth can be very loose. In addition, some queries can be performed on junction trees without actually storing the clique potentials in memory; instead, only the separator potentials are stored. In such a scheme one might want to increase clique size to reduce separator size and, in turn, reduce memory requirements (at the cost of more time) (Dechter and Fattah 2001). In other inference methods clique degree in the underlying junction tree is significant and one might want to enlarge clique size to reduce the junction tree clique degree (Aji and McEliece 2000). Later in this section it will be shown that large cliques can be beneficial when deterministic variables are present in the graph. In all of these cases the treewidth, which is a property of the original graph, can be much smaller than the treewidth of the triangulation that is actually used. In such a case the treewidth of the original graph is not the relevant parameter in determining complexity.

In graphs *without* deterministic variables, state space optimal triangulations might not always be treewidth optimal, but we show here that they will always be within the class of minimal triangulations and therefore obtainable using some elimination order. Many papers attempt to minimize state space by searching over elimination orders (Kjaerulff 1990; Larranaga et al. 1997; Meila and Jordan 1997) implying that the theorem has been assumed in the past, but there does not appear to be a published proof until these authors' work in Bartels and Bilmes (2006). The proof of the following is given in Sect. A.1:

Theorem 4 *Given a graph $G = (V, E)$ where all of the variables are stochastic and have state space ≥ 2, some elimination graph of G will have optimal state space.*

This theorem is important here for two reasons. First, it says that elimination is an effective triangulation search method in the case where state space without determinism is a good indicator of the needed computational resources. In this case one does not need to use the



(a) original untriangulated graph

(b) maximum clique size is 3, state space is $3 \times (2 \times 3 \times 3) = 54$

(c) maximum clique size is 4, state space is $2 \times (3 \times 2 \times 2 \times 2) = 48$

Fig. 3 These graphs demonstrate a triangulation that has optimal state space but is not optimal in maximum clique size (although it is minimal). All variables are stochastic, and the cardinalities are $|a| = |b| = 3$, and $|c_1| = |c_2| = |c_3| = 2$. The difference in state space between the two triangulations can be made arbitrarily large by increasing $|a|$ and $|b|$. *Solid lines* are original graph edges and *dashed lines* are fill-in edges

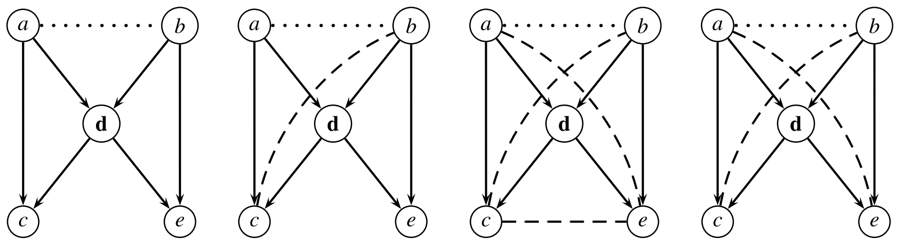
new methods presented later in this paper. Second, the lemmas developed for proving Theorem 4 are used to prove a key theorem in Sect. 6.1. In that section, elimination is used on graphs augmented with certain extra fill-in edges to find state space optimal triangulations for graphs with deterministic variables.

When using deterministic variables, state space optimal triangulations might not be obtainable from any elimination order. Consider Fig. 4 where d is a deterministic function of its parents a and b, the cardinalities of a, b, c and e are all η, and the cardinality of d is $\eta^2 - 1$ (the largest sensible cardinality for d). This graph is triangulated after moralization, and its state space with no additional fill-in is $(2\eta^4 - \eta^2)$. If one considers the graph in Fig. 4(b) the cost is reduced to $(\eta^4 + \eta^3 - \eta^2)$ One might also run elimination beginning with d, resulting in the graph of Fig. 4(c) and cost (η^4). None of these nor any elimination ordering will give the optimal triangulation seen in Fig. 4(d) having state space of $2\eta^3$. This state space is a factor of η smaller than *any* elimination based triangulation, and η can be made arbitrarily large. One might also notice that in this example the problem can be solved by transforming the graph into one that does not include the deterministic variable, where a and b are both connected directly to c and e. Standard elimination can then be used on the transformed graph. Although this approach could work, it will be shown in Sect. 6.1 and Fig. 6 that the optimal choice of which transformations to make cannot be made locally and is not any simpler than choosing a fill-in.

4 Elimination graph detection

It has been demonstrated that elimination is unable to create certain (potentially useful) triangulations, but given a triangulation, how can one tell if an elimination order could have generated it? We give a polynomial time algorithm to solve this problem in Algorithm 1. A correctness proof and an analysis of its complexity is given in Sect. A.2. It takes as input a graph, G, and a triangulation, $T(G)$, and returns true if $T(G)$ can be obtained by some elimination order. This algorithm is essential in our results where we show that most of the desirable triangulations in our test set could not have been generated by elimination.

Next, we illustrate the use of Algorithm 1 on Fig. 4. Consider the graph, G, in Fig. 4(a) and its triangulation, $T(G)$, in Fig. 4(b). First we construct the set $A = \{v|(v \text{ simplicial in } T(G)) \;\&\; (\mathrm{NE}_G(v) = \mathrm{NE}_{T(G)}(v))\}$. The vertices a, c, and e are all simplicial in $T(G)$, but only a and e have the same neighbors in G as they do in $T(G)$. We pick a and eliminate



(a) Original moralized graph (which is already triangulated). Perfect elimination order: c,a,b,d

(b) Elimination triangulation using order: a,c,b,d

(c) Elimination triangulation using order: d,c,a,b

(d) Triangulation not possible with elimination

Fig. 4 The first three plots give examples of triangulations that are obtainable using elimination. The fourth plot is an example of a non-elimination based triangulation with unbounded improvement over all possible elimination based triangulations. *Solid arrows* are original graph edges, *dotted lines* are moralization edges, and dashed lines are fill-in edges. In (**a**), the maximal cliques are (a,b,d), (a,c,d), and (b,d,e) with state spaces of $(\eta \times \eta \times 1)$, $(\eta \times \eta \times (\eta^2 - 1))$, and $(\eta \times (\eta^2 - 1) \times \eta)$ respectively. In (**b**), the maximal cliques are (a,b,c,d) and (b,d,e) with state spaces $(\eta \times \eta \times \eta \times 1)$ and $(\eta \times \eta \times (\eta^2 - 1))$ respectively. In (**c**), there is one maximal clique (a,b,c,d,e) with state space $(\eta \times \eta \times \eta \times 1 \times \eta)$. In (**d**), the maximal cliques are (a,b,c,d) and (a,b,d,e) with state spaces $(\eta \times \eta \times \eta \times 1)$ and $(\eta \times \eta \times 1 \times \eta)$ respectively

Algorithm 1 `isEliminationGraph`

```
On input $\langle G = (V, E), T(G) = (V, E \cup F)\rangle$
if $|V| = 0$ then
  return true
else
  $A = \{v | (v \text{ simplicial in } T(G))\ \&\ (\mathrm{NE}_G(v) = \mathrm{NE}_{T(G)}(v))\}$
  if $A = \emptyset$ then
    return false
  else
    choose $v \in A$, return isEliminationGraph( $G_v$, $(T(G))_v$ )
  end if
end if
```

the node from both graphs. The fill-in edge (b,c) is added to create G_a and no fill-in edges are added to create $(T(G))_a$. We recurse on these elimination graphs, and again we must determine the set A. In this case vertices c and e are simplicial in $T(G)$ and both have the same edges in G and $T(G)$. We eliminate c and recurse. We will stop the example at this point as one can see that b, d, e form a complete graph in both G and $T(G)$, so any ordering of these last three vertices will generate $T(G)$ from G. If one were to record the vertex chosen from A at each step this would give an elimination order that one could use to generate $T(G)$ from G.

For a second example we will run `isEliminationGraph` on the triangulation given in Fig. 4(d). Vertices c and e are simplicial in $T(G)$, but neither vertex has the same edges in G (Fig. 4(a)). Therefore the set A is empty, and the algorithm will return *false* indicating that no elimination order can create $T(G)$ from G.

5 Complexity

It was shown in Sect. 3 that certain large clique and non-minimal triangulations are sometimes desirable. In this section we discuss the computational complexity of finding triangulations in the mixed stochastic/deterministic setting. It was proven in Yannakakis (1981) that it is NP-complete to determine whether a graph has a triangulation with $\leq k$ fill-in edges. In Arnborg et al. (1987) it was shown that it is NP-complete to determine whether a graph has treewidth $\leq k$. The proof in Wen (1991) demonstrated that finding optimal state space triangulations in graphical models with binary variables is NP-hard through a reduction from the Elimination Degree Sequence problem. Here we state that the decision version of the triangulation problem remains in NP for any polynomial time heuristic $f(G, I)$, where I contains vertex information such as cardinality and determinism (i.e., if a vertex is a deterministic function of its parents in the directed graph). This general definition allows us to prove NP-completeness for determining if a triangulation is sufficiently good according to our modified definition of state space. We show that the state space problem remains NP-complete through a reduction from the treewidth problem. This reduction is simpler than the reduction from the Elimination Degree Sequence problem and is valid for variables with arbitrary cardinalities. See Sect. A.3 for the proofs.

Definition 2 Given graph $G = (V, E)$, vertex information I, and threshold α. Define MAXTRI $= \{\langle G = (V, E), I, \alpha\rangle \mid G$ has a triangulation $T(G)$, with $f(T(G), I) < \alpha\}$.

Theorem 5 *The MAXTRI problem is in NP for all polynomial $f(G, I)$.*

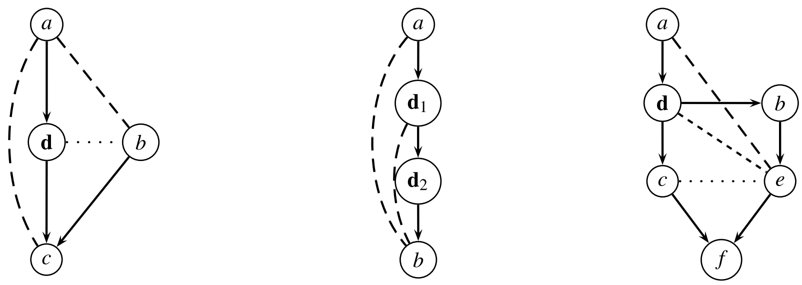
Definition 3 MAXSTATESPACE $= \{\langle G = (V, E), I, \alpha\rangle \mid G$ has a triangulation with state space $< \alpha\}$.

Theorem 6 *MAXSTATESPACE is NP-complete.*

6 Elimination with extra fill-in edges

At this point, we know that elimination alone is not sufficient to generate all triangulations, and more importantly some graphs with deterministic dependencies cannot be optimally triangulated by any elimination order. Because finding state space optimal triangulations is NP-hard, it is necessary to develop heuristic approaches that are able to find triangulations that traditional approaches cannot. This section describes an algorithm called extra-elimination that extends elimination to make it possible to find any triangulation. This algorithm is given here in its most general form, but at this point it has too much flexibility. In this form, the algorithm is therefore not useful other than for illustrative purposes (which is why we describe it in this way). Later in the paper we limit its options to make it a practical search algorithm.

Definition 4 (Extra-Elimination) Alternate the following two steps until no vertices remain: (a) Add edges to the current graph, (b) Eliminate a vertex. When finished, take the union of the extra edges added in the (a) steps and the fill-in edges added in the (b) steps and add them to the original graph.



(a) (a,c) is ancestral from child c, and (a,b) is ancestral because b is a neighbor from moralization

(b) (d_1,b) is ancestral with respect to d_2, (a,b) is ancestral with respect to d_1 because of ancestral edge (d_1,b)

(c) (d,e) is fill-in needed to triangulate the graph, which causes (a,e) to be ancestral

Fig. 5 Illustrations of different types of non-parent neighbors that cause ancestral edges. The deterministic nodes are labeled **d** in each case. *Solid arrows* are original graph edges, *dotted lines* are moralization edges, *long dashed lines* are ancestral edges, and *short dashed lines* are non-ancestral fill-in edges. Figures (**b**) and (**c**) do not show all possible ancestral edges

We formalize this by providing a new definition for a kind of deficiency, and then we regard elimination as a special case of extra-elimination. We have a graph $G = (V, E)$, elimination ordering $\alpha : \{1, 2, \ldots, |V|\} \leftrightarrow V$, and for each $v \in V$ a set of extra edges $H_v \subseteq \{(u_1, u_2) \mid u_1 \neq u_2, (u_1, u_2) \notin E, \alpha^{-1}(v) \leq \alpha^{-1}(u_1), \alpha^{-1}(v) \leq \alpha^{-1}(u_2)\}$, $H = \bigcup_{i=1}^{|V|} H_{\alpha(i)}$.

Definition 5 The **extra-deficiency** of a vertex v in G with extra edges H is: $D_{G,H}(v) = (\{u, w\} \mid \{v, u\} \in E \cup H, \{v, w\} \in E \cup H, \{u, w\} \notin E\})$.

Extra-elimination can now be defined in an analogous manner to elimination. Given $G = (V, E)$, the v**-elimination** graph G_v is defined by adding the edges in H_v and $D_G(v)$ and then deleting v and its incident edges from G. The **elimination graph**, denoted as $\xi_{\alpha,H}(G)$, is the original graph G with the addition of H plus the extra-deficiency edges added from eliminating all variables in the order α. When $H = \emptyset$, extra-deficiency becomes ordinary deficiency, and extra-elimination reduces to elimination. We call an extra edge $e \in H$ **redundant** if $\xi_{\alpha,H}(G) = \xi_{\alpha,H\setminus\{e\}}(G)$. One useful property of this construction comes from the following:

Theorem 7 *Extra-elimination can create any triangulation.*

Note that in this general form we do not have any guidance as to what extra edges to add—this will be addressed in the next section.

6.1 Ancestral pairs

We have presented a general purpose algorithm that can produce arbitrary triangulations, but it is not yet practical—at each elimination step one must choose an arbitrary fill-in, and we have no guidance on how to choose these edges. Extra-elimination can be considerably constrained by taking into account that we are considering Bayesian networks that may have deterministic variables that have values given by arbitrary functions of their parents. In such a graph, moralization will cause each deterministic variable to be in at least one maximal clique with its parents, but the variable might also be a member of other maximal cliques due to its non-parent neighbors. In these cases we can sometimes reduce the state space by adding fill-in edges that ensure that every clique that contains a deterministic variable d also contains d's parents. We call edges that can accomplish this goal *ancestral edges*, and these are the edges that we will pick from when choosing extra fill-in edges.

Definition 6 A pair of nodes (p, c) in a Bayesian network is an **ancestral pair** if and only if p is a parent of a deterministic node d, c is not a parent of d, and $p \neq c \neq d$. An **ancestral edge** is an edge that connects an ancestral pair.

Ancestral edges are named for the case where they connect a node's parent to the node's child, but they can exist for a number of reasons. The first of these is the existence of undirected neighbors due to moralization. An example of this is given in Fig. 5(a). In certain contexts there might be neighbors from other sources as well, such as DBN frame boundaries (Bilmes and Bartels 2003), the creation of cliques for the analysis of posteriors, or other edges that are implied by needing to perform a particular exact probabilistic query. We call ancestral edges that are derived from all of the above cases **original-graph** ancestral edges and the corresponding pairs of nodes **original-graph** ancestral pairs.

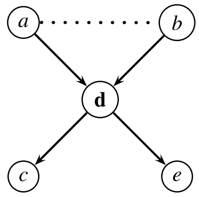
Second, an ancestral edge might obtain its ancestral status only after a different ancestral edge is added to the graph. An example of this is given in Fig. 5(b). Pairs of nodes that are original-graph ancestral, or that become ancestral after other ancestral edges are added are called **pre-triangulation** ancestral pairs. Note that this definition implies a recursion: if an ancestral edge is added it might create additional ancestral pairs. Pre-triangulation ancestral pairs are the basis for the heuristics and results presented in Sects. 6.2 and 7.

Finally, fill-in edges are added to the graph in order to make it triangulated. When a fill-in edge is attached to a deterministic variable it might also form additional ancestral pairs, as in the example given in Fig. 5(c). These along with pre-triangulation ancestral edges make up all possible ancestral edges as given by Definition 6. Unless otherwise stated the theorems in this paper refer to all ancestral edges.

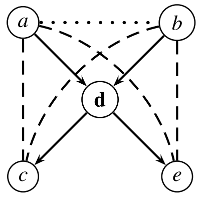
The following theorem states that an optimal state space triangulation can always be found using extra-elimination where we limit the choice of extra edges to ancestral pairs. The proof (in Sect. A.5) shows that the state space optimal triangulation will be a minimal triangulation of a graph augmented by ancestral edges.

Theorem 8 *Elimination with extra-elimination edge addition where the extra edges are limited to ancestral edges is sufficient to find an optimal state space triangulation when all cardinalities are* ≥ 2.

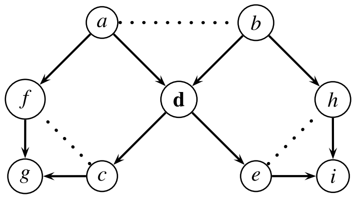
Our problem is still not solved, though. Theorem 8 only tells us that the needed non-minimal edges will be ancestral in the optimal triangulation. Not all ancestral edges may be needed, and not all needed ancestral edges will be known without knowing the rest of the optimal triangulation. It might at first seem that we can look at each deterministic node, d, with parents $\mathrm{pa}(d)$, and non-parent neighbor c and add ancestral edges between $\mathrm{pa}(d)$ and c if $S(c \cup d \cup \mathrm{pa}(d))$ is less than $S(c \cup d) + S(d \cup \mathrm{pa}(d))$, which is a strictly local criterion. For example, in Fig. 6(a) whether or not it is beneficial to add ancestral edges will depend on only the cardinality of d. That is, suppose $|a| = |b| = |c| = |e| = 10$ and $|d| = 40$, then the graph will have a state space of 900 using the triangulated graph with no fill-in that happens to result from moralization. When all ancestral edges have been added as in Fig. 6(b), however, this results in a state space of 2000. On the other hand, if $|d| = 99$,



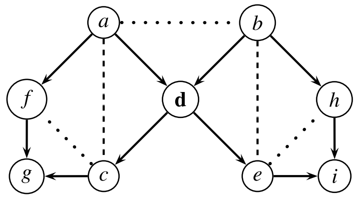
(a) Original Graph



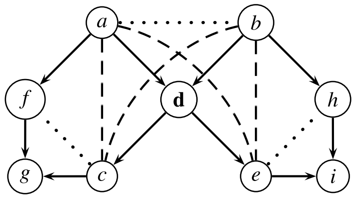
(b) Ancestral edges added to Figure (a)



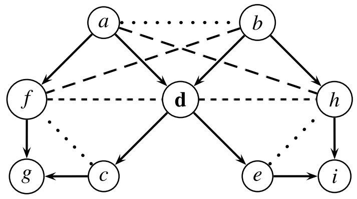
(c) Original (untriangulated) graph



(d) Minimal fill-in



(e) Ancestral edges added



(f) (b, f) and (a, h) are not pre-triangulation ancestral pairs, they become ancestral after adding fill-in edges (d, f), (d, h)

Fig. 6 Graphs illustrating why the choice of ancestral edges cannot be made locally. Variable **d** is deterministic in all graphs. *Solid lines* are original graph edges, *dotted lines* are moralization edges, *long dashed lines* are ancestral edges, and the *short dashed line* is a non-ancestral fill-in edge

then the triangulated by moralization graph has a state space of 2080 while the state space in Fig. 6(b) remains 2000. In this case it is easy to see that the ancestral edges should be added.

In the general case, however, the choice of ancestral edges needed for a state space optimal triangulation cannot be made without knowing the rest of the triangulation. In Fig. 6(c), either (a, c) or (f, d) and either (b, e) or (d, h) needs to be added in order to triangulate the graph. Just as in the Fig. 6(a) $|d| = 40$ case, it is not locally optimal to add the ancestral edges, but if (a, c) and (b, e) are added to triangulate the graph, as in Fig. 6(b), the ancestral edges might now be beneficial. If the cardinality of all of the stochastic variables is 10 and the cardinality of d is 40, then Fig. 6(d) has a state space of 12100, but Fig. 6(e) that includes the ancestral edges has a state space of only 6000.

As mentioned, not all ancestral edges associated with a given deterministic node, d, will be included in the optimal triangulation. In some cases we may leave out ancestral edges in order to take advantage of the factorization provided by d. In other cases we may only add a subset of d's ancestral edges so that it appears with its parents in some maximal cliques and is factorized in others. If we do add all of the ancestral edges associated with d, we can prove that d will only appear in one maximal clique and this clique will include its parents.

Theorem 9 *If there is an edge between all of the parents and non-parent neighbors of a node d, then d will be a member of exactly one maximal clique and this maximal clique will include the parents of d.*

Before moving on we note one possible alternative way to address deterministic variables. Instead of adding ancestral edges, we could transform the graph into one that does not include any of the deterministic variables, and then standard elimination could be used on the transformed graph. In this work, we prefer to work with ancestral pairs. This is because the above transformation does not make the search problem any easier due to the fact that you would not know the complete benefit of a graph transformation without considering the entire resulting triangulation. One would also have to consider the cases where some deterministic variables are kept in the graph for some non-parent neighbors, but transformed away for others. A method could be devised that combines a search over the possible graph transformations and the minimal triangulations of the transformed graphs, but this inhomogeneous process does not have any advantage over the homogeneous process of searching choices of fill-in within the set of ancestral edges. In addition, the deterministic variable might have physical meaning and probabilities based on values of this random variable might be required for a given application domain (see Sect. 7.3)—this would be much more difficult to do in a graph in which the needed node has been completely removed. Lastly, we claim that working with the original non-transformed graph by considering only ancestral edges leads to very simple search heuristics for triangulation that work well as will be demonstrated below.

6.2 Ancestral pair heuristics

Extra-elimination with ancestral edges gives a framework for finding triangulations in networks with deterministic dependencies, but so far we have not given any guidance on which ancestral pairs should be chosen. We now describe a pre-processing step that adds ancestral edges to a graph, and this new graph is then triangulated using standard elimination heuristics. The heuristics that are evaluated here choose edges from the set of pre-triangulation ancestral pairs, and four heuristics are proposed for deciding which extra edges to add:

– **all-extra:** Add all pre-triangulation ancestral edges.
– **sampled-extra:** Randomly select a subset of pre-triangulation ancestral edges.
– **lo-extra:** Choose the pre-triangulation extra edges that are locally optimal. That is, if we have a deterministic node, v, with parents $\mathrm{pa}(v)$ and non-parent neighbor c, the set of edges between c and $\mathrm{pa}(v)$ is locally optimal if $S(c \cup v \cup \mathrm{pa}(v)) < S(c \cup v) + S(v \cup \mathrm{pa}(v))$.
– **some-extra:** Choose all pre-triangulation ancestral edges, with the exception that you ignore ancestral pairs that are a result of undirected edges (such as from moralization). It only considers ancestral pairs resulting from children of deterministic nodes plus any ancestral pairs that are recursively formed from other ancestral edge additions. This method is included primarily to show the effect of not considering these less obvious ancestral edges.

Limiting the edges to pre-triangulation ancestral edges allows the addition to be a pre-processing step that can be followed by any method for finding minimal triangulations. As mentioned, the fill-in from the triangulation step could increase the potential number of ancestral edges, as in Fig. 6(f). For completeness, an algorithm for searching all ancestral pairs is given in Sect. 8.1, but we in this work do not present any further results utilizing it. A genetic algorithm that can generate all possible ancestral edges was presented by these authors in Bartels et al. (2005), Bartels (2008). In some cases, using the genetic algorithm to explore the entire triangulation space gave speedups over the heuristics presented here, but it required initialization using the heuristics.

7 Results

The goal of the experiments is to compare the four extra-elimination heuristics to elimination. The general method for generating a data point for our result set is the following. First, an extra-elimination heuristic and a conventional elimination heuristic are chosen. Next, the chosen methods are used to generate a pool of candidate triangulations. Then, the triangulation in the pool with the smallest state space is selected. This triangulation is used in a calculation of the probability of evidence and the amount of time this calculation takes is measured. This procedure is repeated on a number of combinations of heuristics and pool sizes.

A set of timing measurements is generated for each graph and each of the four extra-elimination heuristics. The procedure used to generate data for the four extra-elimination heuristics is repeated four times using pure elimination. There are two reasons for this repetition. First, it provides a fair comparison between elimination and the overall best of the four extra-elimination methods. Second, it ensures that a large part of the elimination search space has been explored. Note that the elimination triangulations have a significant advantage over the four extra-elimination methods individually.

A variety of state of the art elimination heuristics are used in the results. 20 one-step look ahead heuristics were used, including minimum weight, fill, size, and various combinations and repetitions of these. For each look ahead heuristic there was an additional parameter from 1–3 where the next node in the order is chosen randomly from the top x choices. This parameter is similar to the Stochastic-Greedy Algorithm given in Fishelson and Geiger (2004). Maximum Cardinality Search was also used, bringing the total to 61. For each of the 61 methods, separate pools of 100, 50, 10, and 1 triangulations were created. This was repeated using a modified state space heuristic.

For each graph, the above leads to 488[1] timings for each extra-elimination heuristic, and a total of 19642[2] triangulations generated for each extra-elimination heuristic. There were also $4 \times 488 = 1952$ timings of elimination triangulations and $4 \times 19642 = 78568$ elimination triangulations generated.

As mentioned, the triangulations were timed on a calculation of the probability of evidence. The calculation was stopped if more than 1 gigabyte of memory was used.

All of the timing results were generated using probabilistic inference code that is implemented in highly optimized C++. The inference algorithm itself is a hybrid between the standard Hugin-style inference (Jensen et al. 1990) and the Shenoy-Shafer style inference (Shenoy and Shafer 1986) applied to junction trees, but that exploits sparsity. After a triangulation of the graph has been found, the resultant graph is converted into a junction tree, a

[1](61 triangulation heuristics $\times$ 2 selection heuristics $\times$ 4 pool sizes) = 488 timings.

[2][(61 triangulation heuristics $\times$ 2 selection heuristics) $\times$ pool sizes of $(100 + 50 + 10 + 1)$] = 19642 triangulations.

Table 1 This table compares the performance of the four extra-elimination methods and elimination on randomly generated graphs. The column labeled 'best' gives the number of graphs that the method was the best of all methods. The '$< \times 2$' column gives the number of graphs the method was not the best, but took less than twice the time of the best. The $\times 2$–$\times 4$ column gives the number of graphs where the method was at least twice as slow as the best, but less than four times as slow, and the $\times 4$–$\times 8$ and $\times 8$–$\times 16$ columns are similarly defined. The $>= \times 16$ column gives the count of graphs where the method was more than 16 times as slow as the best (Bartels and Bilmes 2006)

	best	$< \times 2$	$\times 2$–$\times 4$	$\times 4$–$\times 8$	$\times 8$–$\times 16$	$\geq \times 16$
all-extra	240	98	14	1	3	0
sampled-extra	76	186	59	27	5	3
some-extra	50	204	71	21	7	3
lo-extra	18	63	58	68	54	95
elimination	15	43	58	66	56	118

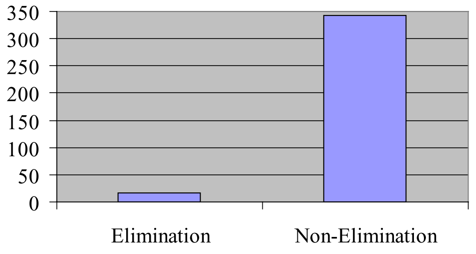


Fig. 7 Number of randomly generated graphs where the fastest triangulation is an elimination graph versus the number not obtainable by any elimination order

tree of cliques that satisfies the running intersection property. A standard "collect-evidence" message order is then determined based on the junction tree, and messages are sent to the root of the tree starting from the leaves. Also, if there is a choice to place a deterministic child or a parent farther from the root along a sub-chain in the tree, preference is given to keep the parent farther from the root—this ensures that sparsity in the model is not computationally costly (as described in the next paragraph).

Each message in the junction tree consists of expanding a clique based on a set of incoming separators and then projecting down to an outgoing separator, where "incoming" and "outgoing" relative to a clique is based on the selected root of the junction tree. Rather than naively expanding the clique, however, we ensure that any sparsity present in the model is never expanded. That is, the actual message expands entries in the clique only that are compatible with the intersection of all incoming separators—this ensures that any entries not in the separator (due to previously discovered zero-scoring or zero-probability entries) do not incur any computational cost. Then, once in the clique, all entries are expanded in topological order w.r.t. the directed Bayesian network (so that a deterministic child is iterated immediately after all of its parents are instantiated). Several additional heuristics are also used, such as a fail-first order (Tsang 1993) and various other standard methods (Dechter 2003). This ensures the preservation of any discovered sparsity during clique expansion. Once done, on projection down to the outgoing separator, of course only the non-zero entries are preserved so that once that separator is used as an incoming separator for some other clique, the benefit of sparsity can propagate throughout the junction tree. Such an approach to inference is quite compatible with the types of triangulations (and resultant junction trees) that our methods produce.

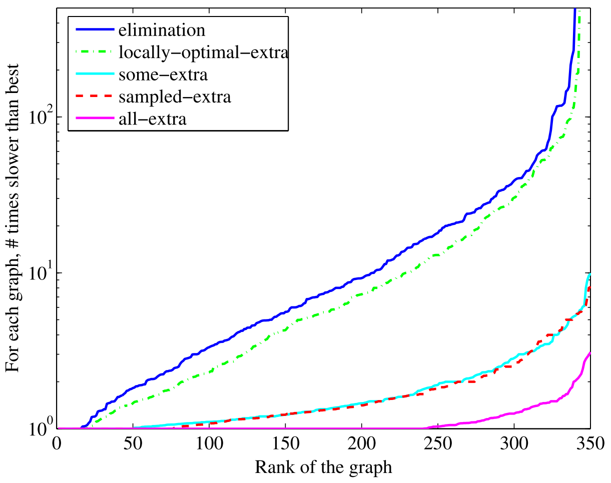


Fig. 8 This figure compares the performance of the four extra-elimination methods and elimination on randomly generated graphs. For each graph, the best time for each method was normalized by the best overall time for the graph. The results for each method were then sorted. The *vertical axis* gives the number of times slower the triangulation was compared to the best for the graph. The *horizontal axis* is an index indicating the nth best graph for the method

7.1 Randomly generated Bayesian networks

The first set of graphs is composed of 356 randomly generated Bayesian networks. Each of these graphs has 30 nodes, a maximum in-degree of 4, and the set of edges is chosen uniformly over all graphs fulfilling the constraints. Each node has a 0.5 probability of being deterministic and a 0.1 probability of being observed. The stochastic variables have cardinalities between 2 and 5 and the observed variables have a cardinality of 50. The deterministic variables have cardinalities between 2 and the product of their parents' cardinalities, with an upper bound of 125. For each graph, the fastest triangulation from each of the five methods is chosen. The 'best' column in Table 1 gives counts of the number of graphs where each method was the best overall. The other columns give the number of graphs where each method was various orders of magnitude slower than the best overall. Figure 7 compares the number of graphs where the best triangulation could and could not have been created using elimination (determined using Algorithm 1). To generate the plot in Fig. 8, for each graph the best time from each method was normalized by the best overall time for the graph. The scores for each method were then sorted. The x axis represents the index of the nth best performing graph for the method, and the y axis represents the number of times slower the method was than the best method for that graph.

In this experiment, all-extra was the overall winner scoring the best on over half of the graphs. Sampled-extra was the second best, followed by some-extra. Lo-extra and elimination performed poorly overall. One would generally expect all-extra to perform well when there is a high percentage of determinism (as in this set of random graphs). Given its strong performance on this task one might even conclude that all-extra is the only method that should ever be considered, but in one case it was 15 times as slow as the fastest triangulation (which was an elimination graph). Sampled-extra has the potential to perform very well as it subsumes all of the other methods, but the large number of fill-in choices keep it, on average, slower than all-extra.

Table 2 gives results over sets of graphs randomly generated the same way as the previous set, but with the exception that there is a fixed number of deterministic variables. The values are the percentage of graphs where that method gave the fastest time or was less than twice the fastest time. This experiment was designed to explore the performance of the heuristics on various classes of graphs. With little determinism (such as the 5 column) there is less

Table 2 Percentage of random graphs with a fixed number of deterministic variables (out of 30) where method gave either the fastest inference time or was $< 2\times$ fastest (Bartels and Bilmes 2006)

# deterministic	5	10	15	20	25
all-extra	82.8%	93.5%	100.0%	87.5%	100.0%
sampled-extra	82.8	87.1	68.3	75.0	100.0
some-extra	62.1	71.0	70.7	75.0	100.0
lo-extra	44.8	51.6	17.1	50.0	89.3
elimination	58.6	35.5	9.8	54.2	71.4

opportunity for improvement over elimination and we see smaller improvements. With a lot of determinism (such as the 25 column) the total state space of the graph is small and the solution can be found quickly regardless of the triangulation, so again there is less differentiation. All-extra again out performed the other methods, and gave an especially large advantage over the other methods in the 15 column.

7.2 Randomly generated dynamic Bayesian networks

The next set of graphs is 10 randomly generated dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1989). DBNs are composed of a graph template that is repeated to match the length of a particular observed sequence. Triangulation is performed on a subgraph that is created from the template rather than on the much larger, unrolled graph (Kjærulff 1992; Xiang 1999; Murphy 2002). This subgraph is typically one instance of the template along with a subset of variables from a second instance, together called the 1.5 slice (Murphy 2002). This additional set, called the interface, contains the variables that are directly connected to any variable in an adjacent template instance. The interface sets (on both sides of the template) are completed so that an unrolled triangulation can be constructed by piecing together these triangulated subgraphs. It was noted in Xiang (1999), Bilmes and Bartels (2003) that the template as given by the graph designer might not form the most optimal structure for the purposes of triangulation and inference. All the graphs in this work are re-templated using the boundary algorithm described in Bilmes and Bartels (2003). This method searches the graph template for an interface set that has a lower cost than the interface set as defined by the original template. The triangulations performed here are done on an instance of the 1.5 slice of the re-templated graph. During inference, there is one instance of the template for each observation, and the template instances are “glued” together at the interface cliques. Each template instance is known as a **frame**.

The random DBNs have 24 vertices per frame, a maximum in-degree of 3, and the set of edges is chosen with uniform probability over all graphs fulfilling the constraints. Each node has a 0.5 probability of being deterministic and a 0.2 probability of being observed. The stochastic variables have cardinalities between 2 and 5 and the observed variables have a cardinality of 50. The deterministic variables have cardinalities between 2 and the product of their parents’ cardinalities, with an upper bound of 125.

In addition to the set of triangulations and timings described at the beginning of the section, a set of 488 timings were generated using one instance of maximum cardinality search, minimum weight, fill, or size (labeled **Once**). This is to compare our elimination baseline to a more typical triangulation baseline.

The results are split across Tables 3 and 4. Figure 9 presents the results in the same manner as Fig. 8. The numbers are the average amount of time in seconds to compute the probability of evidence per 100 frames (that is, per 100 instances of the triangulated template). The amount of time spent timing each triangulation is fixed for each graph, and multiple

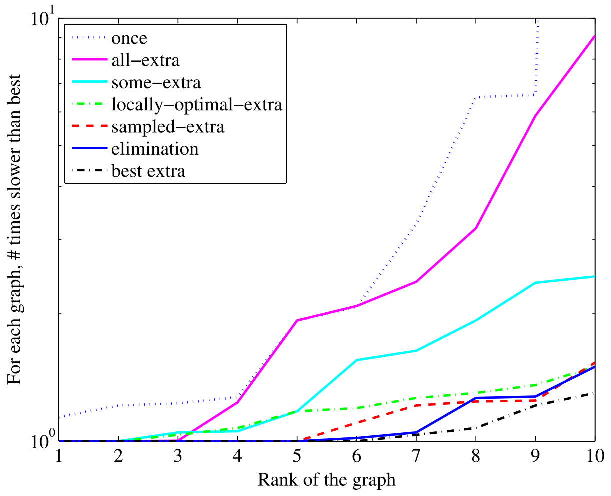


Fig. 9 This figure compares the performance of the four extra-elimination methods and elimination on randomly generated graphs. For each graph, the best time for each method was normalized by the best overall time for the graph. The results for each method were then sorted. The *vertical axis* gives the number of times slower the triangulation was compared to the best for the graph. The *horizontal axis* is an index indicating the *n*th best graph for the method. 'best-extra' is the best of all-extra, some-extra, locally-optimal-extra, and sampled-extra

observation sequences might be evaluated during that time. Longer times were chosen for slower graphs that gave poor statistics in preliminary runs. A triangulation fails if it uses more than 1 gigabyte of memory or fails to calculate the probability of evidence of 1 frame during the fixed time window.

A 25%–31% improvement was seen on 3 graphs, a nominal improvement was seen on 1 graph, 3 cases gave comparable results, and elimination did better than the edge addition heuristics on 3. These DBN results were much less dramatic than on the "non-dynamic" networks from the previous section. This is primarily because the interface cliques can account for a majority of the compute time. These cliques can also make the graphs fairly dense to begin with, leaving the resulting space of possible triangulations smaller than on graphs with less structure.

As a group, the extra-elimination algorithms outperformed elimination, but the large pool of elimination triangulations did better than any individual method. Sampled-extra was the best of the extra-elimination algorithms, while all-extra was the worst. This difference from the previous set of graphs is likely explained by the many edge additions needed to complete the interface cliques. These edges give sampled-extra a smaller set of options to explore, and because the interface edges make the graphs somewhat dense all-extra can be too aggressive. The "once" set of triangulation performed poorly demonstrating the effectiveness of the large variety of elimination and selection heuristics present in the other methods.

Tables 3 and 4 also give statistics on the mean, variance, and median times as well as the percentage of timings that did not complete without running out of memory. Note that for some graphs the variance can be quite high (Graph 5, some-extra has a mean of 253.9 seconds with a variance of 1094.2), and for others the failure rate is significant (Graph 1, some-extra has a failure rate of 90.3%). This is because the differences between triangulations can give exponential differences in cost. Recall that the state space of a clique is the product of the state spaces of its variables. The Once heuristics explore a smaller portion of the triangulation space than the other methods and can have a much higher failure rate.

7.3 Example real world graph

Our own interest in these networks arises from the use of DBNs to represent speech recognition and natural language processing systems. Such systems give explicit representations of

Table 3 This table gives the first half of the results for the randomly generated DBNs. The second half of the results is in Table 4. All scores are based on the amount of time in seconds to compute the probability of evidence per 100 observations (that is, per 100 instances of the triangulated template). '% Fail' gives the percentage of triangulation attempts where the time or memory limit was reached before the probability of evidence frame could be computed for one frame. '$\mu \pm \sigma$' is the mean and variance of the (non-failing) times. The median is over all attempts, and if more than half of the triangulations failed no median is given. Finally, 'Best' indicates the smallest time for all triangulation attempts. Various ratios of times are given, and the last row tells if elimination could have produced the best (or indistinguishably close to best) overall triangulation

		Graph 1	Graph 2	Graph 3	Graph 4	Graph 5
Once	% Fail	%93.6	%99.0	%30.9	%2.0	%41.2
	$\mu \pm \sigma$	518.6 ± 8.7	108.5 ± 1.7	189.0±197.9	25.9 ± 75.4	257.6 ± 203.9
	Median	–	–	201.4	3.2	452.3
	Best	511.0	106.7	39.7	1.7	31.6
Elim.	% Fail	%71.9	%29.4	%10.8	%6.5	%28.7
	$\mu \pm \sigma$	849.5 ± 234.6	659.6 ± 622.9	338.2±443.1	65.6 ± 246.7	187.3 ± 212.1
	Median	–	1287.0	136.9	3.1	203.1
	Best	469.9	**51.4**	**32.3**	**1.4**	20.9
All	% Fail	%88.5	%11.9	%9.4	%6.8	%78.5
	$\mu \pm \sigma$	622.9 ± 114.2	203.9 ± 75.0	297.6±280.7	34.9 ± 173.3	985.0 ± 764.6
	Median	–	182.0	184.8	3.0	–
	Best	**449.0**	122.4	102.9	2.7	149.1
Samp.	% Fail	%83.1	%38.3	%12.2	%8.8	%51.8
	$\mu \pm \sigma$	974.3 ± 474.6	349.6 ± 368.8	375.8±395.1	70.0 ± 223.8	406.2 ± 401.5
	Median	–	568.2	215.0	4.7	–
	Best	**448.0**	64.1	40.1	1.7	**16.4**
Some	% Fail	%90.3	%27.3	%6.6	%3.1	%44.1
	$\mu \pm \sigma$	655.0 ± 137.0	283.2 ± 263.7	477.1±356.4	84.2 ± 215.2	253.9 ± 1094.2
	Median	–	198.3	533.0	27.2	443.7
	Best	473.0	84.1	50.2	2.7	17.2
L.O.	% Fail	%70.5	%38.5	%15.6	%5.1	%33.2
	$\mu \pm \sigma$	865.1 ± 199.8	531.9 ± 493.8	533.9±499.4	105.1 ± 258.1	170.2 ± 202.7
	Median	–	893.7	657.5	13.1	299.9
	Best	536.5	53.2	34.7	1.9	19.3
Best once/Best extra		1.14	2.00	1.14	0.99	1.92
Median elim./all		–	7.07	0.74	1.03	–
Median elim./samp.		–	2.27	0.64	0.66	–
Median elim./some		–	6.49	0.26	0.11	0.46
Median elim./L.O.		–	1.44	0.21	0.24	0.68
Best elim./Best extra		1.05	0.97	0.93	0.85	1.28
Best Is Elim.		no	yes	yes	yes	no

Table 4 This table gives the second half of the results for the randomly generated DBNs. The first half of the results along with an explanation of the table is given in Table 3

		Graph 6	Graph 7	Graph 8	Graph 9	Graph 10
Once	% Fail	%0.2	%1.0	%46.9	%100.0	%1.2
	$\mu \pm \sigma$	64.7 ± 99.2	61.6 ± 66.8	146.4±119.9	–	76.2 ± 27.5
	Median	1.4	42.5	394.0	–	86.4
	Best	1.3	35.2	22.4	–	18.6
Elim.	% Fail	%2.4	%7.8	%25.7	%88.6	%3.6
	$\mu \pm \sigma$	21.8 ± 80.8	136.4 ± 177.9	222.2±304.6	195.2 ± 263.8	60.4 ± 95.6
	Median	1.6	56.7	202.2	–	35.0
	Best	**0.2**	**27.7**	4.3	**0.3**	**5.8**
All	% Fail	%0.2	%22.5	%3.9	%86.7	%1.4
	$\mu \pm \sigma$	14.8 ± 99.0	291.6 ± 165.7	39.5±122.4	15.8 ± 61.1	27.8 ± 5.7
	Median	0.4	229.0	27.9	–	30.7
	Best	**0.2**	162.8	4.2	**0.2**	11.9
Samp.	% Fail	%1.2	%18.3	%16.4	%78.0	%6.4
	$\mu \pm \sigma$	13.0 ± 51.6	436.5 ± 419.2	238.5±412.0	314.9 ± 469.6	47.2 ± 95.8
	Median	0.6	390.9	94.1	–	23.5
	Best	**0.2**	42.5	**3.4**	**0.2**	6.3
Some	% Fail	%0.0	%24.1	%6.8	%81.5	%77.9
	$\mu \pm \sigma$	10.8 ± 86.0	752.3 ± 466.2	69.0±180.2	10.3 ± 25.9	186.5 ± 259.6
	Median	0.4	1178.0	39.7	–	–
	Best	**0.2**	67.9	4.0	**0.2**	13.5
L.O.	% Fail	%2.3	%11.1	%50.0	%86.9	%12.9
	$\mu \pm \sigma$	11.5 ± 96.4	310.5 ± 208.8	61.1±145.6	156.5 ± 211.8	86.7 ± 291.5
	Median	0.4	308.4	–	–	82.7
	Best	**0.2**	36.0	4.3	0.3	**5.7**
Best once/Best extra		6.70	0.98	6.57	–	3.27
Median elim./all		3.93	0.25	7.26	–	1.14
Median elim./samp.		2.49	0.14	2.15	–	1.48
Median elim./some		3.85	0.05	5.10	–	–
Median elim./L.O.		3.67	0.18	–	–	0.42
Best elim./Best extra		1.00	0.77	1.27	1.28	1.02
Best Is Elim.		yes	yes	no	no	yes

the relationships between words, syllables, phones, or parts of speech. A very basic model for sequences is called the *triangle structure* and can be seen in Fig. 10. This structure was introduced in Zweig (1998) and can be used to solve the following problem. Assume we have a known sequence of symbols, and we have a sequence of observations that is significantly longer than the symbol sequence. The example symbol sequence we will use here is "c"—"a"—"t" (modeling the word "cat"). We also have a sequence of 100 observations

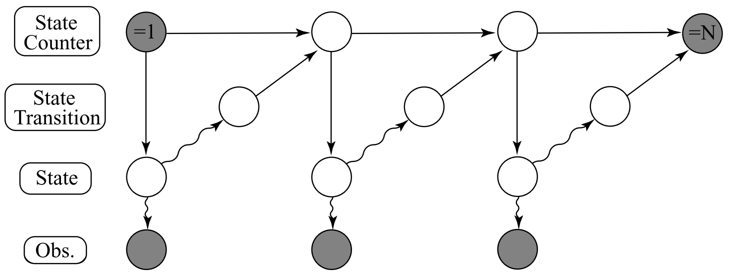


Fig. 10 Triangle Structure (Zweig 1998), can be used for aligning a fixed length sequence to acoustic observations. *White* variables are hidden and *shaded* are observed. *Straight arrows* represent deterministic relationships, and *wavy arrow* represent probabilistic relationships. This figure shows 3 "frames", but in practice the number of frames will depend on the length of the observation sequence (in speech graphs there are typically hundreds of frames)

representing audio data. We know that the audio data contains the word "cat", but we would like to find out which observations correspond to "c", which correspond to "a", and which correspond to "t". The graph for this problem has four types of variables, and a variable of each type is repeated once for each observation. They are as follows:

– *State Counter*: Deterministic variable indicating the position in the sequence. In the "cat" example it counts from 1 to 3.
– *State*: Deterministic variable that gives the identity of the current state in the sequence. In the example it takes on the values "c", "a", and "t".
– *State Transition*: Boolean variable indicating if the model is in the last frame of the current state. This is a probabilistic variable with a distribution over "true" and "false".
– *Observation* (*Obs.*): Vector of acoustic observations.

The *State Counter* is set to 1 in the first frame and set to the 3 in the last frame. When the *State Transition* from the previous frame is false, *State Counter* copies its value from the *State Counter* in the previous frame. When the *State Transition* from the previous frame is true, *State Counter* is set to 1 larger than the *State Counter* from the previous frame. The *State* variable gives the identity of the current phone in the sequence by mapping 1 to "c", 2 to "a", and 3 to "t". *State* is necessary because in many words the same phone appears more than once and different values of *State Counter* might map to the same state. The *Observation* is conditioned on *State* to give a different output distribution for each phone. In the figure wavy arrows indicate that $p(\mathit{Obs.}|\mathit{State})$ and $p(\mathit{State\ Transition}|\mathit{State})$ are probabilistic. Straight arrows show that $p(\mathit{State}|\mathit{State\ Counter})$ and $p(\mathit{State\ Counter}|\mathit{Previous\ State\ Counter})$ are deterministic.

The triangle structure also forms the basis for more complicated graphs. Figure 11 gives an example of how it can be built upon to create a speech recognizer. In this graph the identity of the word or words is chosen probabilistically. The triangle structure still models a deterministic sequence of phones, but the choice of sequence is randomly selected through the choice of word. The variables in the graph are as follows:

– *Word*: A variable representing the identity of the current word.
– *Word Transition*: Boolean variable indicating if the model is in the last frame of the current word.
– *State Counter*, *State Transition*, *State*, *Observation* (*Obs.*): These four variables form the triangle structure, just as in Fig. 10.

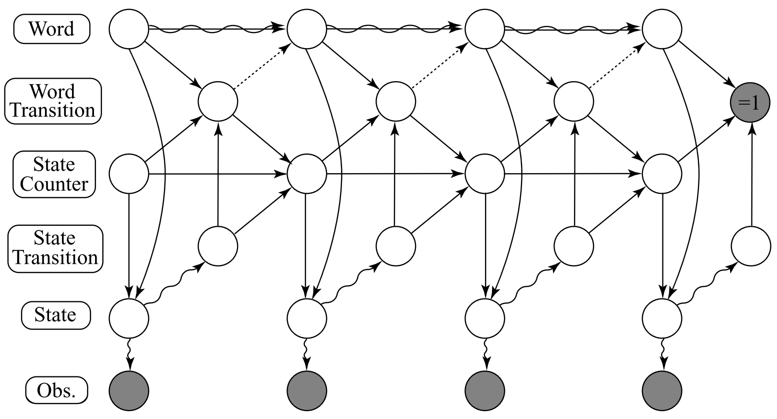


Fig. 11 Dynamic Bayesian Network for speech recognition (Bilmes et al. 2001). *Wavy arrows* are probabilistic dependencies, *solid, straight arrows* are deterministic dependencies, and dotted arrows are switching dependencies. *Shaded nodes* are observed variables, *unshaded nodes* are hidden variables

When the *Word Transition* is "true" a new value of *Word* is chosen probabilistically. This is based on a distribution that is conditioned on the previous value of *Word*. When *Word Transition* is "false" the value of the *Word* variable from the previous frame is simply copied into the current frame. *Word Transition* is a "switching parent" to *Word* because its value changes the type of conditional probability table that *Word* uses. This is indicated by the dotted arrow from *Word Transition* to *Word*. There are both straight and wavy arrows between the *Word* variables since this relationship can be either deterministic or random. *Word Transition* itself is always deterministic. It is "true" when *State Counter* has iterated through the entire sequence for the current word and "false" otherwise. The triangle structure behaves as before, but with two modifications so that it can handle more than one word. First, *State* is conditioned on *Word* as well as *State Counter* so that its value can change for different words. Second, *State Counter* has *Word Transition* as a parent so that the count can be reset to 0 whenever the end of a word is reached.

The third speech recognition graph is given in Fig. 12. This graph uses two symbol sequences that are allowed to be asynchronous. An example use for this graph is a speech recognizer that uses both audio data of a person's voice and video data of a person's lip movements. Lip movements tend to anticipate the voice so it can be useful to model audio and video with separate symbol sequences that are allowed to start and end at different times. This graph differs from Fig. 11 primarily in that it has two triangle structures. The first triangle is composed of the variables *State Counter 1*, *State Transition 1*, *State 1* and *Observation* (*Obs.*) *1*. The second is composed of the variables *State Counter 2*, *State Transition 2*, *State 2* and *Observation* (*Obs.*) *2*. The other difference is that *Word Transition* is only "true" when both triangle structures reach the end of their sequences.

7.4 Real world graphs

The final set of graphs is the following 10 real-world dynamic Bayesian networks:

- **Aurora Decoding**—whole word model for speech recognition, Fig. 11 (Bilmes et al. 2001).
- **Edit Distance training 1, 2, decoding**—learns edit distance parameters from data (Filali and Bilmes 2005).

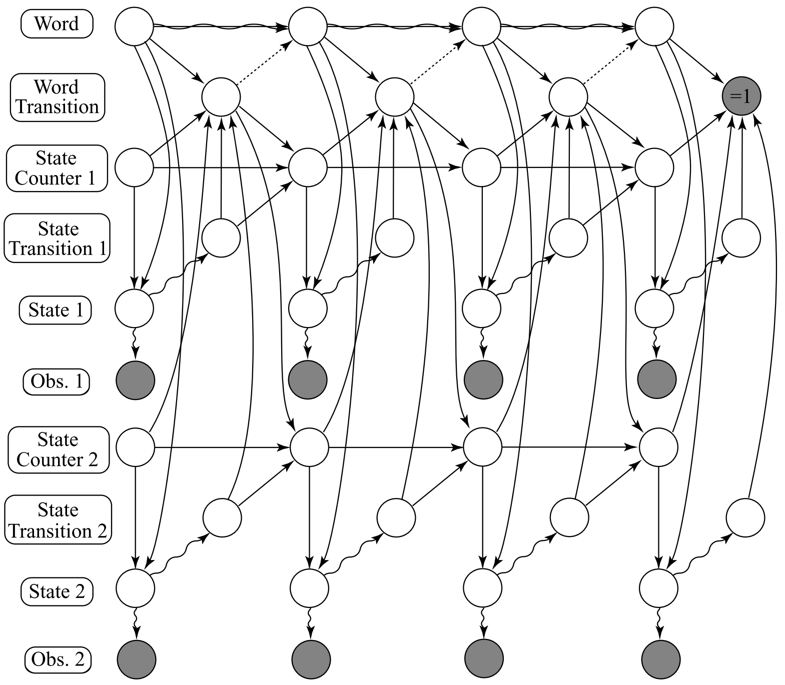


Fig. 12 Dynamic Bayesian Network for speech recognition using two synchronous symbol sequences. Similar to the graph given in Zhang et al. (2003). *Wavy arrows* are probabilistic dependencies, *solid, straight arrows* are deterministic dependencies, and *dotted arrows* are switching dependencies. *Shaded nodes* are observed variables, *unshaded nodes* are hidden variables

- **Feature Detect**—extracts phonetic features from speech data (courtesy of Simon King).
- **Image Concept Detect**—for image classification (courtesy of Brock Pytlik).
- **Mandarin**—speech recognition graph modeling Mandarin Chinese tonal phones using asynchronous spectral and pitch feature streams (Lei et al. 2005).
- **MultiStream**—speech recognition training graph with asynchronous feature streams, Fig. 12, based on the graphs used in Zhang et al. (2003) and Subramanya et al. (2004).
- **PhoneFree 1, 2**—isolated word pronunciation scoring using a phone-free model (courtesy of Karen Livescu).

The set of triangulation methods and timings are the same used for the randomly generated DBNs. The real-world graphs were also re-templated in the same manner, and the results are again the average amount of time in seconds to compute the probability of evidence per 100 frames. The results are given in Tables 5 and 6, and the range of results for each method can be seen in Fig. 13.

On the real-world DBNs, significant improvement was seen on 4 of 10 graphs with 2 of these more than doubling in speed. These improvements were larger than what was seen on the randomly generated DBNs. One reason is that the interfaces on real-world graphs tend to be smaller on these human-designed graphs than on the randomly generated interfaces.

On this set of graphs, locally-optimal-extra performed very well on its own. Recall that it performed poorly on the non-dynamic randomly generated graphs and gave average per-

Table 5 This table gives results for real-world DBNs. All scores are based on the amount of time in seconds to compute 100 DBN frames. '% Fail' gives the percentage of triangulation attempts where the time or memory limit was reached before the probability of evidence frame could computed for one frame. '$\mu \pm \sigma$' is the mean and variance of the (non-failing) times. The median is over all attempts, and if more than half of the triangulations failed no median is given. Finally, 'Best' indicates the smallest time for all triangulation attempts. Various ratios of times are given, and the last row tells if elimination could have produced the best (or indistinguishably close to best) overall triangulation (Bartels and Bilmes 2006)

		Aurora Decode	Edit Dist. Train 1	Edit Dist. Train 2	Edit Dist. Decode	Feature Detect
Once	% Fail	%0.0	%2.0	%20.3	%17.8	%0.0
	$\mu \pm \sigma$	0.9 ± 0.1	36.5 ± 36.1	39.2 ± 13.1	1240.0 ± 2714.8	618.5 ± 293.1
	Median	0.9	36.0	36.1	96.5	576.4
	Best	0.8	11.0	30.6	24.4	378.8
Elim.	% Fail	%0.0	%5.9	%8.3	%4.7	%22.3
	$\mu \pm \sigma$	1.3 ± 5.1	18.5 ± 79.7	83.4 ± 294.6	264.4 ± 1144.4	495.9 ± 222.7
	Median	0.8	11.7	40.5	28.8	527.7
	Best	**0.2**	**2.8**	**4.6**	**1.0**	12.2
All	% Fail	%0.0	%8.8	%85.0	%88.7	%39.8
	$\mu \pm \sigma$	0.4 ± 1.3	11.4 ± 35.9	57.5 ± 366.1	221.8 ± 784.7	646.9 ± 216.7
	Median	0.2	6.6	–	–	1011.5
	Best	**0.2**	**2.8**	6.2	**1.0**	212.9
Samp.	% Fail	%0.0	%6.0	%34.4	%27.3	%61.1
	$\mu \pm \sigma$	1.4 ± 10.1	11.5 ± 9.8	356.3 ± 1065.6	387.7 ± 1145.8	464.6 ± 518.9
	Median	0.2	10.6	256.7	193.4	–
	Best	**0.2**	**2.8**	4.8	**1.0**	18.2
Some	% Fail	%0.0	%3.9	%66.0	%66.2	%8.8
	$\mu \pm \sigma$	0.3 ± 0.9	19.9 ± 133.1	28.9 ± 52.2	117.6 ± 462.1	108.3 ± 192.0
	Median	0.2	6.8	–	–	31.1
	Best	**0.2**	**2.8**	6.2	**1.0**	12.7
L.O.	% Fail	%0.0	%4.1	%55.7	%55.5	%30.5
	$\mu \pm \sigma$	0.4 ± 1.4	9.1 ± 3.9	298.1 ± 1272.3	102.4 ± 387.0	140.7 ± 267.0
	Median	0.2	7.3	–	–	130.1
	Best	**0.2**	**2.8**	**4.5**	**1.0**	**8.9**
Best once/Best extra		5.02	3.94	6.75	25.14	42.39
Median elim./all		4.17	1.77	–	–	0.52
Median elim./samp.		3.80	1.10	0.16	0.15	–
Median elim./some		4.30	1.72	–	–	16.94
Median elim./L.O.		3.77	1.59	–	–	4.06
Best elim./Best extra		1.16	1.00	1.01	1.00	1.37
Best Is Elim.		no	yes	yes	yes	no

Table 6 This table gives the second half of the results for the randomly generated DBNs. The first half of the results along with an explanation of the table is given in Table 5 (Bartels and Bilmes 2006)

		Image Detect	Mandarin	Multi-Stream	Phone Free 1	Phone Free 2
Once	% Fail	%0.0	%42.6	%0.0	%0.0	%25.4
	$\mu \pm \sigma$	44.0 ± 1.6	12.9 ± 0.2	6.4 ± 2.6	4.9 ± 0.4	34.6 ± 10.9
	Median	43.5	13.1	5.6	5.0	45.1
	Best	41.9	12.6	5.2	3.5	9.8
Elim.	% Fail	%4.9	%34.0	%3.8	%15.7	%18.4
	$\mu \pm \sigma$	48.6 ± 49.9	12.8 ± 10.1	20.4 ± 118.7	138.5 ± 1062.8	106.8 ± 1081.5
	Median	42.4	12.9	5.5	4.8	29.6
	Best	**28.5**	**6.8**	**3.0**	2.2	4.2
All	% Fail	%74.8	%3.3	%1.2	%80.3	%54.1
	$\mu \pm \sigma$	48.5 ± 86.9	8.4 ± 5.5	9.1 ± 59.0	768.7 ± 2749.0	517.9 ± 2020.7
	Median	–	7.6	3.9	–	–
	Best	29.2	7.4	3.6	3.7	**1.9**
Samp.	% Fail	%22.4	%12.7	%4.2	%44.7	%53.5
	$\mu \pm \sigma$	102.4 ± 63.7	10.1 ± 8.9	19.2 ± 71.2	1822.1 ± 2737.6	2145.6 ± 3351.7
	Median	123.7	9.6	5.7	4633.9	–
	Best	29.1	**6.8**	**3.0**	6.5	33.4
Some	% Fail	%2.9	%4.1	%2.6	%35.5	%36.5
	$\mu \pm \sigma$	327.8 ± 228.8	8.3 ± 3.6	34.2 ± 75.4	165.8 ± 1371.0	169.9 ± 1607.4
	Median	356.0	7.6	10.7	14.3	27.7
	Best	29.2	7.4	**3.0**	2.5	3.5
L.O.	% Fail	%9.2	%2.5	%0.8	%38.1	%38.1
	$\mu \pm \sigma$	45.7 ± 20.2	8.0 ± 3.4	5.3 ± 14.0	61.1 ± 262.2	23.2 ± 4.7
	Median	42.9	7.0	3.6	39.2	23.9
	Best	**28.8**	**6.8**	**2.9**	**1.0**	2.4
Best once/Best extra		1.46	1.86	1.78	3.57	5.11
Median elim./all		–	1.70	1.41	–	–
Median elim./samp.		0.34	1.34	0.96	0.00	–
Median elim./some		0.12	1.70	0.51	0.34	1.07
Median elim./L.O.		0.99	1.84	1.51	0.12	1.24
Best elim./Best extra		0.99	1.00	1.02	2.29	2.21
Best Is Elim.		yes	yes	yes	no	no

formance on the randomly generated DBNs. One explanation for this is that many of the real-world graphs have deterministic variables with much larger cardinalities than the surrounding random variables. Another is that the logical structure in the real world graphs favors the more obvious locally optimal triangulations.

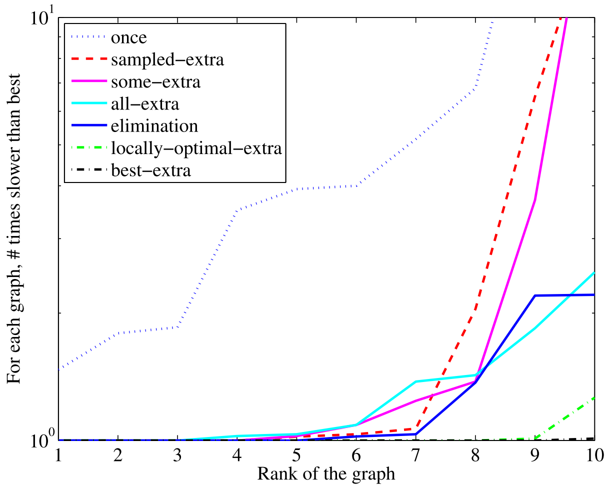


Fig. 13 This figure compares the performance of the four extra-elimination methods and elimination on randomly generated graphs. For each graph, the best time for each method was normalized by the best overall time for the graph. The results for each method were then sorted. The *vertical axis* gives the number of times slower the triangulation was compared to the best for the graph. The *horizontal axis* is an index indicating the nth best graph for the method. 'best-extra' is the best of all-extra, some-extra, locally-optimal-extra, and sampled-extra. 'best-extra' has a value of $1(10^0)$ until its tenth graph

7.5 Discussion

These results demonstrate empirically that non-minimal triangulations produced with the extra-elimination heuristics can give significant performance gains over state of the art elimination triangulations. The various extra-elimination heuristics gave different results for different tasks. They had, by far, the largest effect on the first set of randomly generated graphs. Only small improvements were seen on the randomly generated DBNs, but significant gains were seen on the real-world DBNs. In practice the best results will be obtained by using a variety of heuristics and comparing their performance.

8 Alternative methods for mixed stochastic/deterministic graph triangulation

Extra-elimination with pre-triangulation ancestral edges is just one possible way to produce non-minimal triangulations. In this section we present some alternatives.

A scheme for producing a class of non-minimal triangulations, called super-cluster-trees, was introduced in Dechter and Fattah (2001) for the purpose of finding junction trees with small separator sizes. This algorithm collapses adjacent junction tree nodes that have large separators until the maximum size of the separators falls to an acceptable level. This algorithm, although useful for its intended purpose of minimizing separator size, cannot form all possible triangulations. For example, the triangulation in Fig. 4(d) cannot be formed by merging the junction tree nodes from the triangulations in Figs. 4(a), 4(b), or 4(c). Similarly, one can reduce separator size using a super-bucket-tree created by eliminating a group of variables at once (Dechter 2003). This algorithm can also create some non-minimal triangulations but cannot be used to create Fig. 4(d). Certain junction tree cliques were also combined in Olesen and Madsen (2002) for the purpose of finding the maximal prime subgraph decomposition of Bayesian networks, but this decomposition was intended for use in approximate inference, a pre-processing step for divide and conquer triangulation, or incremental junction tree construction.

One could find optimal triangulations by adding extra edges to a graph after a triangulation is produced by elimination. The disadvantage of this is that adding these edges might cause the graph to become untriangulated. The super-cluster-tree algorithm is an instance of post-elimination edge addition and could be extended with additional junction tree manipulations to create arbitrary triangulations (a similar algorithm was proposed in Draper 1995 for forming triangulations). To devise such an algorithm one would have to consider operations that cover all desirable triangulations, keep the graph triangulated, and take into consideration that there might be many junction trees for a given triangulation.

Another alternative is to begin with a seed triangulation and then refine it by changing it one edge at a time. This scheme is based on the following theorem (Lauritzen 1996; Bartels et al. 2005):

Theorem 10 *Given any two triangulations of a graph, $T_a(G)$ and $T_b(G)$ there is a sequence of triangulated graphs $T_a(G), T_1(G), T_2(G), \dots, T_b(G)$ with only a single edge difference between subsequent graphs in the sequence.*

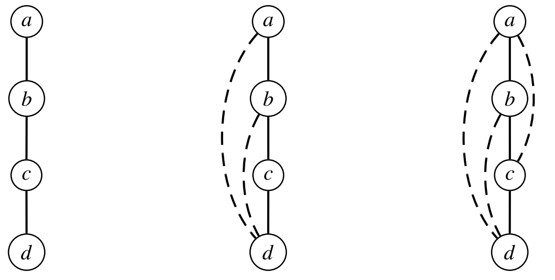
This was used for a genetic search by these authors in Bartels et al. (2005), but could be used as a basis for many types of stochastic search. One could also use this concept in a heuristic search that begins by adding all possible fill-in edges and then removes them until some stopping condition is reached, or a method that begins with elimination and adds edges one at a time.

8.1 Node connected extra-elimination

Next, we present an Extra-Elimination based algorithm that we call **Node Connected Extra-Elimination**. Extra-elimination allows extra edges to be added at any point during the elimination process, but the method given in Sect. 6.2 only added edges before any elimination was performed. Alternatively, one could eliminate several vertices and then add edges to the remaining graph, eliminate more vertices, and repeat. The following theorem states that the same triangulation will result regardless of when the extra edges are added (with the restriction that one is not allowed to add extra edges to a vertex that has already been eliminated).

The following theorem defines this idea more precisely by defining a sequence of extra edge sets $H_{\alpha(i)}, i = 1 \dots |V|$. The theorem says that you can move edges between the sets $H_{\alpha(i)}$ to form a new sequence of extra edges $\mathscr{H}_{\alpha(i)}$ $i = 1 \dots |V|$ as long as: 1) an extra edge is not connected to a node that has already been eliminated; 2) the two sequences contain the same set of non-redundant ancestral edges. The basis of the proof is that when one eliminates a node v the deficiency only depends on the neighbors of v (with respect to both the original graph edges and new edges in H that are adjacent to v), so an extra edge will not effect the elimination fill-in until one of the vertices it is connected to is eliminated. The full proof is given in Sect. A.4.

Theorem 11 *We are given a graph $G = (V, E)$, elimination ordering α, sets of extra edges $H_{\alpha(i)}, i = 1 \dots |V|$ with $H = \bigcup_{i=1}^{|V|} H_{\alpha(i)}$, extra-elimination graph $\xi_{\alpha,H}(G) = (V, E \cup H \cup F)$, and redundant extra edges $H_r \subseteq H$. We define a new set of extra edges $\mathscr{H}_{\alpha(i)}$ $i = 1 \dots |V|$ with $\mathscr{H} = \bigcup_{i=1}^{|V|} \mathscr{H}_{\alpha(i)}$ and with redundant edges $\mathscr{H}_r$ such that $H \setminus H_r = \mathscr{H} \setminus$*



(a) Original graph which is already triangulated.

(b) Elimination based non-minimal triangulation

(c) Optimal triangulation, not obtainable by elimination.

Fig. 14 Example of non-minimal triangulation for an undirected graph. Each edge in the graph represents the constraint that the variables it connects must be equal. *Solid lines* are original graph edges and *dashed lines* are fill-in edges

$\mathscr{H}_r$ and all edges in $\mathscr{H}$ satisfy the definition of an extra edge with respect to graph $G' = (V, E \cup F)$ (that is $\forall\,(u_1, u_2) \in \mathscr{H}_v,\ u_1 \neq u_2, (u_1, u_2) \notin E \cup F, \alpha^{-1}(v) \leq \alpha^{-1}(u_1), \alpha^{-1}(v) \leq \alpha^{-1}(u_2)$). Under such conditions, it is the case that $\xi_{\alpha,H}(G) = \xi_{\alpha,\mathscr{H}}(G)$.

This theorem is the basis for **Node Connected Extra-Elimination**. At each elimination step, one chooses a node to eliminate and a (possibly empty) set of extra edges that are connected to the node. This algorithm will never consider adding extra edges that are not connected to the vertex that is about to be eliminated. The proof of the following is given in Sect. A.4:

Theorem 12 *Node Connected Extra-Elimination can produce any triangulation.*

The following theorem tells us that all of the extra edges added during **Node Connected Extra-Elimination** will be non-minimal in the final graph. This is a guarantee that if additional edges are added the search will span the space of non-minimal triangulations. Again, the proof is in Sect. A.4.

Theorem 13 *In extra-elimination, if an extra edge is added to a node just before the node is eliminated then the extra edge will be non-minimal. In other words, all $(u, v) \in H_v$ are non-minimal.*

If we limit the extra edges to those that are ancestral, it follows directly from Theorem 8 that the state space optimal triangulation is in its search space. It follows directly from Theorem 11 that the algorithm can generate any triangulation that can be generated using the pre-triangulation heuristics from Sect. 6.2.

9 Undirected models

So far we have discussed how non-minimal triangulations and ancestral edges can be useful for improving the efficiency of inference in directed networks. This concept can easily be extended to undirected networks. Undirected models can generalize directed models, and all of the previous discussion applies directly when a variable in an undirected model is a deterministic function of a subset of its neighbors. Although such a situation is plausible, symmetric constraints between neighbors are more typical in undirected models. A simple example is a constraint that forces a set of variables to be equal. Similarly, one might have a constraint that a set of N particular variables must sum to a constant. In this case, knowing $N-1$ of the variables in the relationship will allow you to determine the value of the last variable.

Suppose we are given a constraint over a set of variables, $X_1, \ldots, X_n$. If it is the case that knowing the values of some of the variables in $X_1, \ldots, X_n$ gives us the value of the other variables, then all of the variables in $X_1, \ldots, X_n$ can be seen as being deterministic. All of the variables in $X_1, \ldots, X_n$ also play an analogous role to parent variables in the directed case. Recall that ancestral edges in directed graphs connect the parents of a deterministic variable D to non-parent neighbors of D. Similarly, we can say that an ancestral edge is one that connects a variable in $X_1, \ldots, X_n$ to a neighbor of a different variable in $X_1, \ldots, X_n$. Next, we give definitions for deterministic variable and ancestral edge that work for both directed and undirected models.

A more general definition of a deterministic variable is as follows. Suppose we have a probability distribution of the form:

$$p(X_G) = \frac{1}{Z} \prod_{C_i \in \mathscr{C}} f_{C_i}(X_{C_i})$$

$$Z = \sum_{X_i \in X_G} \sum_{x \in X_i} \prod_{C_i \in \mathscr{C}} f_{C_i}(x_{C_i})$$

Where X_G is the set all variables, C_i are clique functions, and X_{C_i} is the set of variables in the clique C_i. A variable $D \in X_G$ is deterministic if there exists a function $f_{C_i}(D, X_1, \ldots, X_n)$ where $f_{C_i}(D = d_k, X_1 = x_1, \ldots, X_n = x_n) \neq 0$ and $f_{C_i}(D \neq d_k, X_1 = x_1, \ldots, X_n = x_n) = 0$ for all assignments to the variables $X_1 = x_1, \ldots, X_n = x_n$. The value of $D = d_k$ can be different for different assignments to $X_1, \ldots, X_n$.

In the directed case, $X_1, \ldots, X_n$ are the parents of D and f_{C_i} is a conditional probability table that equals 1 for values of D that are consistent with a deterministic function of D's parents and 0 elsewhere. Note that when the function is one-to-many this definition does not result in the parent variables being named as deterministic. The function f_{C_i} can also be a symmetric constraint. For example, if we are given a constraint that says that $X_1 = X_2 = X_3$ then f_{C_i} is a function that is non-zero for assignments where the constraint is met and zero elsewhere. The definition would mark all three of X_1, X_2, and X_3 as deterministic.

We can now generalize the definition of ancestral edge. Given the function $f_{C_i}(D, X_1, \ldots, X_n)$ that defines the variable D as deterministic and a set of edges E defined by the clique functions of $p(X_G)$, the ancestral edges are:

$$\big\{(U, V) \in E \mid (U, D) \in E, U \in \{X_1, \ldots, X_n\}, (D, V) \in E, V \notin \{X_1, \ldots, X_n\}\big\}$$

In the directed case, $X_1, \ldots X_n$ are the parents of D, and this definition states that U is a parent of D and V is a non-parent neighbor of D. This is the same as in Definition 6. Note that this now generalizes to symmetric constraints in undirected models. Given the example where $X_1 = X_2 = X_3$, an edge that connects X_1 to a neighbor of X_2 is ancestral. In this case X_2 is deterministic and X_1 plays the role of "parent".

The use of ancestral edges for undirected graphs is illustrated in Fig. 14. In this graph, each pair of variables connected by an edge is constrained to be equal. Once we know one of the variable values, we know the other three with probability 1. Suppose the state space of each variable is η. In the original graph given by Fig. 14(a) we have three maximal cliques. Because of the constraints, each maximal clique has a state space of η giving a total state space of 3η. In Fig. 14(b) we have a triangulation that is obtainable by elimination that reduces this to 2η. Finally, Fig. 14(c) gives the optimal triangulation with state space η which is not obtainable using elimination. In this example of symmetric constraints the improvement is a function of the number of variables in the graph. In the undirected case the improvement from making use of ancestral edges was unbounded, as in Fig. 4.

10 Conclusion

This paper has shown that large clique triangulations can be computationally useful on graphs containing deterministic variables. An example was given where the optimal triangulation has a state space that is arbitrarily smaller than all elimination based triangulations. An algorithm was presented to determine if a triangulation could have been generated using elimination, and it was shown that the generalized triangulation problem is NP-complete. Extra-elimination was introduced as a framework for producing any triangulation, and it was proven that extra edges can be limited to the ancestral edges when optimizing for state space. Novel heuristics based on ancestral edges were presented and results were given on randomly generated and real world graphs. It was also shown how non-minimal triangulations can be applied to undirected models with symmetric constraints. Future work will include a joint search for triangulation and within-clique dynamic variable orderings for use in hybrid inference/search procedures.

Acknowledgements This work was supported by ONR MURI grant N000140510388 and by NSF grant IIS-0093430.

Appendix

A.1 Elimination is state-space optimal in positive graphs

Lemma 1 *Let $G=(V,E)$ and $G'=(V,E')$ both be triangulated graphs with $E'\subseteq E$ and $|E\setminus E'|=k$. Then there is an increasing sequence $G'=G_0\subset\cdots\subset G_k=G$ of triangulated graphs that differ by exactly one edge* (Lauritzen 1996, *Lemma* 2.21, *p.* 20)

Lemma 2 *Consider a graph $G=(V,E)$ with any two distinct maximal cliques C_1 and C_2. There exists nodes w_1 and w_2 such that $w_1\in C_1$, $w_2\in C_2$, $w_1\notin C_2$, $w_2\notin C_1$, and $(w_1,w_2)\notin E$.*

Proof Suppose the lemma is not true, then all $w_1\in C_1\setminus C_2$ are connected to all $w_2\in C_2\setminus C_1$. Because C_2 is maximal, there cannot exist $v\in V\setminus C_2$ such that $\forall\, w_2\in C_2$, $(w_2,v)\in E$. This implies that $C_1\setminus C_2=\emptyset$, and we have a contradiction because C_1 is also a maximal clique and since distinct must have at least one vertex that is not in C_2. □

Lemma 3 *Consider a non-minimally triangulated graph $T(G)=(V,E\cup F)$, i.e., where an edge $(u,v)\in F$ exists such that $T'(G)=(V,E\cup(F\setminus\{(u,v)\}))$ is also triangulated. There is only one maximal clique in $T(G)$ containing both u and v.*

Proof Assume the contrary, so there is more than one maximal clique in $T(G)$ containing u and v. We choose two distinct maximal cliques C_1 and C_2 both containing u and v, and choose vertices w_1 and w_2 such that $w_1 \in C_1$, $w_2 \in C_2$, $w_1 \notin C_2$, $w_2 \notin C_1$, and $(w_1, w_2) \notin E \cup F$. There will be a cycle in $T'(G)$, $u - w_1 - v - w_2 - u$, and because $T'(G)$ is triangulated this cycle must have a chord. Nodes w_1 and w_2 are not connected so u and v must be connected, but this is a contradiction because $T'(G)$ does not contain edge (u, v). □

Lemma 4 *Consider a non-minimally triangulated graph $T(G) = (V, E \cup F)$, i.e., where there exists an edge (u, v) such that $T'(G) = (V, E \cup (F \setminus \{(u, v)\}))$ is also triangulated. Say that C is the* unique *maximal clique in $T(G)$ that contains both u and v and all variables in this clique are stochastic (i.e., not deterministic). In $T'(G)$, C will be split into the cliques $C_u = C \setminus \{u\}$ and $C_v = C \setminus \{v\}$. Define c as the state space of the clique $C \setminus \{u, v\}$. If C_u and C_v are maximal cliques in $T'(G)$ the difference in state space $S(T'(G)) - S(T(G))$ will be $c(|u| + |v| - |u||v|)$. If C_u is a subset of another maximal clique in $T'(G)$ the difference will be $(1 - |u|)c|v|$, if C_v is a subset of another maximal clique, it will be $(1 - |v|)c|u|$, and if both are subsets of their respective maximal cliques it will be $-c|u||v|$.*

Proof The cliques not containing both u and v will be unaffected by the edge removal. The state space of C is $c|u||v|$, and the state spaces of C_v and C_u (if they exist) are $c|v|$ and $c|u|$ respectively. □

Lemma 5 *Given a graph $G = (V, E)$ where all variables are stochastic (i.e., not deterministic) with cardinality ≥ 2, a triangulation that is state space optimal and minimal will exist.*

Proof Suppose we have a non-minimal triangulation $T(G)$ and remove an edge (u, v) creating new triangulation $T'(G)$. If either of the new cliques created by the edge removal are not maximal in $T'(G)$ the difference in state space will be negative (from Lemma 4 and because $|u| \geq 2$ and $|v| \geq 2$). When both cliques are maximal, the state space difference is $c(|u| + |v| - |u||v|)$, again negative. From Lemma 1, when considering the increasing sequence in reverse, we can create a decreasing sequence of graphs from any non-minimal triangulation to some minimal triangulation, and the state space of each graph in the sequence will be less than or equal to the previous graph. Hence, any non-minimal triangulation will have a spanning subgraph that is a minimal triangulation with a smaller or equal state space. □

Theorem 4 *Given a graph $G = (V, E)$ where all of the variables are stochastic and have state space ≥ 2, some elimination graph of G will have optimal state space.*

Proof By Lemma 5 a triangulation exists that is state space optimal and minimal, and by Theorem 3 it is achievable using elimination. □

A.2 `isEliminationGraph` (Algorithm 1)

Theorem 14 *Let $G = (V, E)$ be a graph, α be an elimination ordering, and $\xi_\alpha = (V, E \cup F)$ be the graph where G is eliminated according to α. Then $(v, w) \in E \cup F$ if and only if there exists a chain $[v, v_1, v_2, \ldots, v_k, w]$ in G such that $\alpha^{-1}(v_i) < \min(\alpha^{-1}(v), \alpha^{-1}(w))\ \forall i = 1 \ldots k$* (Rose et al. 1976, *Lemma* 4).

The following lemma is assumed by Ohtsuki et al. (1976) but never proven, and a similar lemma for triangulated graphs and strict inclusion is given in Rose (1970, Lemma 3).

Lemma 6 *Consider graph $G_A = (V, E)$ which is a spanning subgraph of $G_B = (V, E \cup E_B)$. If both graphs are eliminated according to ordering α creating $\xi_\alpha(G_A) = (V, E \cup F_A)$ and $\xi_\alpha(G_B) = (V, E \cup E_B \cup F_B)$, then $F_A \subseteq (E_B \cup F_B)$.*

Proof The lemma is clearly true for one vertex. Now assume it is true for G_A, G_B, and ordering α with N vertices, and consider graphs $G_{A'}, G_{B'}$, and ordering α' all with $N+1$ vertices. These graphs can be defined as $\xi_{\alpha'}(G_{A'}) = (V \cup \alpha'(1), E \cup E_{A'} \cup F_A \cup D_{G_{A'}}(\alpha(1)))$ and $\xi_{\alpha'}(G_{B'}) = (V \cup \alpha'(1), E \cup E_B \cup E_{B'} \cup F_B \cup D_{G_{B'}}(\alpha(1)))$ where $E_{A'} = \{(u, v) \in G_{A'} : v = \alpha(1)\}$ and $E_{B'} = \{(u, v) \in G_{B'} : v = \alpha(1)\}$.

$G_{A'}$ is a spanning subgraph of $G_{B'}$ so by definition $(E \cup E_{A'}) \subseteq (E \cup E_B \cup E_{B'})$. By definition we also know that $\mathrm{NE}_{G_{A'}}(\alpha(1)) \subseteq \mathrm{NE}_{G_{B'}}(\alpha(1))$, so we can conclude that $D_{G_{A'}}(\alpha(1)) \subseteq D_{G_{B'}}(\alpha(1))$. From the induction hypothesis we know that $F_A \subseteq F_B$. Therefore, $(E \cup E_{A'} \cup F_A \cup D_{G_{A'}}(\alpha(1))) \subseteq (E \cup E_B \cup E_{B'} \cup F_B \cup D_{G_{B'}}(\alpha(1)))$. □

We supply one more lemma before proving the correctness of `isEliminationGraph`. This lemma shows that in the case where $T(G)$ is an elimination graph of G, a node is simplicial in $T(G)$, and the node has the same neighbors in G and $T(G)$, then we can chose this node as the first node of an elimination order.

Lemma 7 *We are given a graph $G = (V, E)$ and a triangulation $T(G) = (V, E \cup F)$. Suppose F can be generated by some elimination order α, and where there exists k such that $\alpha(k)$ is simplicial in $T(G)$ and $\mathrm{NE}_G(\alpha(k)) = \mathrm{NE}_{T(G)}(\alpha(k))$. Then, the order β with $\beta(1) = \alpha(k), \beta(2) = \alpha(1), \beta(3) = \alpha(2), \dots, \beta(k) = \alpha(k-1), \beta(k+1) = \alpha(k+1), \dots, \beta(|V|) = \alpha(|V|)$ will also generate F.*

Proof Consider eliminating nodes in the already triangulated graph $T(G)$ according to order β. $\alpha(k)$ is simplicial in $T(G)$ so eliminating it will not add edges to $T(G)$. Moreover, $\alpha(1)$ is simplicial in $T(G)$, so it will still be simplicial in $(T(G))_{\alpha(k)}$. Continuing with the nodes in β in turn, for each $i = 2, \dots, k-1, k+1, \dots, |V|$, $\alpha(i)$ will not have any neighbors in $T(G)$ that it does not also have at the corresponding point when, according to α, it is eliminated from $T(G)$. Therefore, β is a perfect ordering of $T(G)$. Define $\xi_\beta(G) = (V, E \cup F_\beta)$ where F_β are the fill-in edges produced when eliminating G according to β. G is a spanning subgraph of $T(G)$ so if we eliminate both $T(G)$ and G according to β, from Lemma 6, $F_\beta \subseteq F$.

We wish next to show that $F_\beta = F$. Suppose to the contrary that there is some edge $(v, w) \in F$ but $(v, w) \notin F_\beta$. This edge cannot be adjacent on the node $\alpha(k)$ (meaning $v, w \neq \alpha(k)$) because $\mathrm{NE}_G(\alpha(k)) \subseteq \mathrm{NE}_{\xi_\beta(G)}(\alpha(k)) \subseteq \mathrm{NE}_{T(G)}(\alpha(k)) = \mathrm{NE}_G(\alpha(k))$. Define $S_\alpha = \{u \in V : \alpha^{-1}(u) < \min[\alpha^{-1}(v), \alpha^{-1}(w)]\}$ and $S_\beta = \{u \in V : \beta^{-1}(u) < \min[\beta^{-1}(v), \beta^{-1}(w)]\}$. Clearly, if $k < \min[\alpha^{-1}(v), \alpha^{-1}(w)]$ then $S_\alpha = S_\beta$, and if $k > \min[\alpha^{-1}(v), \alpha^{-1}(w)]$ then $S_\alpha = S_\beta \setminus \{\alpha(k)\}$. From Theorem 14, there is a path in G, $[v, v_1, v_2, \dots, v_l, w]$, where $\alpha(v_i) \in S_\alpha, \forall i = 1 \dots l$. But since $(v, w) \notin F_\beta$, no such path $[v, v_1, v_2, \dots, v_m, w]$ exists in G such that $\beta(v_i) \in S_\beta, \forall i = 1 \dots m$. This, however, is a contradiction since $S_\alpha \subseteq S_\beta$. Therefore, we must have that $(v, w) \in F_\beta$. Therefore, $F_\beta = F$. □

Theorem 15 `isEliminationGraph` *will return true if and only if F can be generated by some elimination order α.*

Proof First we will show that if the algorithm returns *true* then $\exists\ \alpha$ such that $\xi_\alpha = T(G)$. The proof is by induction on $|V|$. The theorem obviously holds for $|V| = 1$, assume it is true when $|V| = N - 1$, and consider a graph where $|V| = N$. The first node chosen by the algorithm is v. We have assumed that the algorithm returns *true* on the pair $\langle G, T(G)\rangle$, so it must also return *true* on $\langle G_v, (T(G))_v\rangle$. From the induction hypothesis there exists $\xi_{\beta(G_v)} = (T(G))_v$ so we construct elimination ordering α by concatenating v to the front of ordering β. $\mathrm{NE}_G(v) = \mathrm{NE}_{T(G)}(v)$ so $\mathrm{NE}_{\xi_\alpha(G)}(v) = \mathrm{NE}_{T(G)}(v)$. From the induction hypothesis, the same holds for all other vertices, $V \setminus \{v\}$.

Next we show that if there is some elimination order α that generates F, or $\xi_\alpha = (V, E \cup F)$, then `isEliminationGraph` will return *true*. This will also be proven by induction on $|V|$. The theorem obviously holds for $|V| = 1$, assume it is true when $|V| = N - 1$, and consider a graph where $|V| = N$. From Lemma 7, we can create an order β where some node, $\alpha(k)$, that is simplicial in $T(G)$ and $\mathrm{NE}_G(\alpha(k)) = \mathrm{NE}_{T(G)}(\alpha(k))$ is eliminated first. Note that $\alpha(1)$ will be simplicial in $T(G)$ and $\mathrm{NE}_G(\alpha(1)) = \mathrm{NE}_{T(G)}(\alpha(1))$, so at least one $\alpha(k)$ will always exist. When the elimination graph $G_{\alpha(k)}$ is eliminated in the order β without considering $\alpha(k)$ it will generate $T(G)_{\alpha(k)}$, and from the induction hypothesis `isEliminationGraph` will return true. □

We conclude this section with an analysis of the complexity of `isElimination-Graph`. The graphs G and $T(G)$ can be defined using adjacency matrices. First, consider the complexity of a single call to the algorithm. There are three tasks that might depend on the size of the input graph: (1) finding the simplicial nodes of $T(G)$, (2) determining if a simplicial node has the same neighbors in G and in $T(G)$, (3) eliminating the node from the graph. As a first step we create a list of maximal cliques that exist in $T(G)$. This list is useful because a node is simplicial if and only if it is a member of exactly one maximal clique. Each maximal clique can be represented by an array of binary indicators for each node in the graph. Using maximal cardinality search (Tarjan and Yannakakis 1984) to find a perfect ordering, the list of maximal cliques can be created in $O(|V|^2)$ (this can also be done in $O(|V| + |E|)$ time using a Fibonacci heap). An $O(|V|)$ loop is used to examine the vertices for membership in A. In practice one only needs to find a single vertex in A, but all of the vertices might need to be examined before finding it. Nested inside this loop are the two checks for membership in A. To determine if the current node is simplicial one must check its membership in $O(|V|)$ maximal cliques. One can check that the node has the same neighbors in G and $T(G)$ in $O(|V|)$ by iterating through a row in the adjacency matrices. To eliminate the chosen node, first we connect its neighbors in G which takes $O(|V|^2)$ time, and then we build new adjacency matrices which takes $O(|V|^2)$ time. All three tasks are $O(|V|^2)$, and we recurse $O(|V|)$ times which brings the complexity to $O(|V|^3)$.

A.3 MAXSTATESPACE is NP-complete

Theorem 5 *The MAXTRI problem is in NP for all polynomial $f(G, I)$.*

Proof The following program can verify a member of MAXTRI with a choice of fill-in F:

$V =$“On input $\langle V, E, I, F, \alpha\rangle$:

1. Build the graph $G = (V, E \cup F)$
2. Check if G is triangulated
3. If $f(G, I) < \alpha$ *accept*; otherwise *reject*”

The size of F is $< |V|^2$, testing if $G = (V, E \cup F)$ is triangulated can be done in polynomial time (Rose et al. 1976; Tarjan and Yannakakis 1984), and testing that $f(G, I) < \alpha$ is polynomial time by definition. □

Definition 7 MAXTREEWIDTH $= \{\langle G = (V, E), k\rangle \mid G$ has treewidth $\leq k\}$.

Theorem 16 *MAXTREEWIDTH is NP-complete* (Arnborg et al. 1987).

Theorem 6 *MAXSTATESPACE is NP-complete.*

Proof First it is shown that MAXTREEWIDTH is polynomial time reducible to MAXSTATESPACE. A graph with tree-width k has a maximum clique size of $k + 1$ and when it has no deterministic variables and all variables are of the same cardinality $|v|$, the state space of the maximum clique will be $|v|^{k+1}$. Suppose we are given an instance of MAXTREEWIDTH with graph G and parameter k. We construct an I assigning each vertex in G a cardinality of $|V|$ (i.e., the cardinality of each node is equal to the number of nodes in the graph), and define no vertex as being deterministic. If the tree-width of G is at most k, since there are no more than $|V| - 1$ maximal cliques in G, the graph cannot have a state-space larger than $(|V| - 1)|V|^{k+1}$. If on the other hand the tree-width of G is strictly larger than k, the smallest possible state space for G would be $|V|^{k+2} > (|V| - 1)|V|^{k+1}$. Therefore, the tree-width of G is at most k if and only if MAXSTATESPACE(G, I, $|V|^{k+2}$) accepts.

Next, the maximal cliques of a triangulated graph can be found in polynomial time (Gavril 1972, Sect. 2), and given the maximal cliques the state space can be calculated in polynomial time, so this satisfies Theorem 5, which implies that MAXSTATESPACE is in NP. Therefore MAXSTATESPACE is NP-complete. □

A.4 Extra elimination

Theorem 7 *Extra-elimination can create any triangulation.*

Proof To obtain an arbitrary $T(G) = (V, E \cup F)$, add F as extra edges and eliminate according to some perfect elimination order. □

Theorem 11 *We are given a graph $G = (V, E)$, elimination ordering α, sets of extra edges $H_{\alpha(i)}, i = 1 \ldots |V|$ with $H = \bigcup_{i=1}^{|V|} H_{\alpha(i)}$, extra-elimination graph $\xi_{\alpha,H}(G) = (V, E \cup H \cup F)$, and redundant extra edges $H_r \in H$. We define a new set of extra edges $\mathscr{H}_{\alpha(i)}$ $i = 1 \ldots |V|$ with $\mathscr{H} = \bigcup_{i=1}^{|V|} \mathscr{H}_{\alpha(i)}$ and with redundant edges $\mathscr{H}_r$ such that $H \setminus H_r = \mathscr{H} \setminus \mathscr{H}_r$ and all edges in $\mathscr{H}$ satisfy the definition of an extra edge with respect to graph $G' = (V, E \cup F)$ (that is $\forall\ (u_1, u_2) \in \mathscr{H}_v,\ u_1 \neq u_2, (u_1, u_2) \notin E \cup F, \alpha^{-1}(v) \leq \alpha^{-1}(u_1), \alpha^{-1}(v) \leq \alpha^{-1}(u_2)$). Under such conditions, it is the case that $\xi_{\alpha,H}(G) = \xi_{\alpha,\mathscr{H}}(G)$.*

Proof The theorem will be proven by induction on $|V|$. It is obviously true for $|V| = 1$, assume it is true for $|V| = N - 1$, and consider the case where $|V| = N$. We define the sets $H_1 = \{(u, v)|(u, v) \in H_{\alpha(1)}, u = \alpha(1)\}$ and $H_{\neq 1} = H_{\alpha(1)} \setminus H_1$. The edges in H_1 cannot be redundant because $\alpha(1)$ is the first node eliminated and cannot gain any neighbors except through $H_{\alpha(1)}$. We have that $H_1 \subseteq \mathscr{H}_{\alpha(1)}$ because $H \setminus H_r = \mathscr{H} \setminus \mathscr{H}_r$ and one cannot add edges to $\alpha(1)$ after it is eliminated (if an edge in H_1 was in some other $\mathscr{H}_{\alpha(i \neq 1)}$ this would not be a legal extra edge). When $\alpha(1)$ is eliminated the fill-in edges that will be added depend only on $NE_G(\alpha(1))$ and H_1, so we have $NE_{\xi_{\alpha,H}}(\alpha(1)) = NE_{\xi_{\alpha,\mathscr{H}}}(\alpha(1))$. The fill-in created from the elimination of $\alpha(1)$ will not be effected by edges in $H_{\neq 1}$, so we can move all of $H_{\neq 1}$ from $H_{\alpha(1)}$ to $H_{\alpha(2)}$. To see this note that we will have an equivalent graph if we add $H_{\alpha(1)}$, eliminate $\alpha(1)$, and add $H_{\alpha(2)}$, or if we add $H_{\alpha(1)} \setminus H_{\neq 1}$, eliminate $\alpha(1)$, and add $H_{\alpha(2)} \cup H_{\neq 1}$. Note that some edge in $H_{\neq 1}$ might be added by the elimination of $\alpha(1)$, but such edge is redundant and can simply be removed from H. Similarly we can move $\mathscr{H}_{\alpha(1)} \setminus H_1$ from $\mathscr{H}_{\alpha(1)}$ to $\mathscr{H}_{\alpha(2)}$. After adding H_1 and eliminating $\alpha(1)$ we have a graph with $|V| = N - 1$ vertices and our newly formed H,$\mathscr{H}$ meet the conditions of the theorem, so we can use the induction hypothesis to complete the proof. □

Theorem 12 *Node Connected Extra-Elimination can produce any triangulation.*

Proof One can produce any triangulation by adding all of the fill-in edges and then eliminate using a perfect elimination order. From Theorem 11 we will get the same triangulation regardless of if the extra edges are added before any nodes are eliminated or if they are added at later steps of Extra-Elimination. □

Theorem 13 *In extra-elimination, if an extra edge is added to a node just before the node is eliminated then the extra edge will be non-minimal. In other words, all $(u, v) \in H_v$ are non-minimal.*

Proof Suppose extra edge (u, v) chords some four cycle v, n_1, u, n_2, v. We know that $\alpha^{-1}(v) < \alpha^{-1}(u)$ because (u, v) cannot be added after u is eliminated, and $\alpha^{-1}(v) < \alpha^{-1}(n_1)$, $\alpha^{-1}(v) < \alpha^{-1}(n_2)$ or (u, v) would exist from the elimination of n_1 or n_2 and could not be in H_v. When v is eliminated it is already neighbors with n_1 and n_2 because it cannot gain any neighbors after it is eliminated. The elimination of v will connect n_1 and n_2 and the cycle will be chorded without (u, v). □

A.5 Ancestral edges

Lemma 8 *Consider a graph G with node cardinalities ≥ 2 and triangulation $T(G)$. Suppose an edge (p, c) is added to form triangulation $T_{new}(G)$. If $S(T_{new}(G)) < S(T(G))$ then (p, c) is ancestral with respect to some deterministic node D.*

Proof From Lemma 3 there is only one maximal clique, C in $T_{new}(G)$ that contains both p and c. In $T(G)$, there will be maximal cliques C_p, C_c such that $p \in C_p$ and $c \in C_c$. Since the state space of cliques not containing p or c will not be changed by the edge addition we only consider C, C_p, and C_c. All three of C, C_p, C_c cannot all be maximal cliques in $T_{new}(G)$ because $S(T_{new}(G)) < S(T(G))$. Therefore, one or both of C_p, C_c are not maximal cliques in $T_{new}(G)$, and one of C_p and C_c have a larger state space than C. Without loss of generality, we say that $S(C_c) < S(C)$. Because $C_c \subset C$ and $S(C_c) > S(C)$, we know there must be some deterministic variable D such that $D \in S_c$. We also know that there is some variable set P such that $p(D = d, P_1 = P_1, \dots, P_n = p_n) = 1$. It must be the case that $P \setminus C_c \neq \emptyset$ and $P \setminus C = \emptyset$ or it would not be the case that $S(C_c) > S(C)$. Since $C = C_p \cup \{p\}$ we know that $p \in P$. We know that $c \notin P$ or else (p, c) would already be part of G. We know that $(p, D) \in E \cup F$ and $(D, c) \in E \cup F$ since $p, D, c \in C$. Therefore, D is ancestral in $T_{new}(G)$. □

Theorem 8 *Elimination with extra-elimination edge addition where the extra edges are limited to ancestral edges is sufficient to find an optimal state space triangulation when all cardinalities are ≥ 2.*

Proof of Theorem 8 We are given a graph $G = (V, E)$ and we wish to obtain optimal state space triangulation $T(G) = (V, E \cup F)$. Define set A as all edges that are in F and ancestral in $T(G)$. Next, construct $G' = (V, E \cup A)$. $T(G)$ is a triangulation of G' with fill-in $F' = F \setminus A$. We will next show that F' is a minimal triangulation of G', and because minimal triangulations can always be created using elimination we can then conclude that $T(G)$ can be formed by adding A to G followed by elimination. Assume that this conclusion is not true, and there is a set of edges $M \subset F'$ such that $T_{min}(G') = (V, (E \cup A) \cup M)$ is triangulated. From Lemma 1 there is an increasing sequence of graphs that differ by one edge beginning with $T_{min}(G')$ and ending with $T(G)$. State space $S(T(G))$ is optimal so $S(T_{min}(G')) \geq S(T(G))$. If $S(T_{min}(G')) > S(T(G))$ at least one graph in the sequence of graphs from $T_{min}(G')$ to $T(G)$ must have a lower state space than the previous graph. From Lemma 8, this is a contradiction because F' does not contain any ancestral edges. Therefore, F' is a minimal triangulation of G' and $T(G)$ is an elimination graph of G'. If $S(T_{min}(G')) = S(T(G))$ we have not shown a contradiction but $T_{min}(G')$ is also an optimal triangulation and is an elimination graph of G'. □

Theorem 9 *If there is an edge between all of the parents and non-parent neighbors of a node d, then d will be a member of exactly one maximal clique and this maximal clique will include the parents of d.*

Proof We prove that d will be in one maximal clique by contradiction. Suppose d appears in more than one maximal clique, two of these being C_1 and C_2. Since the cliques are maximal, there must be v_1, v_2 such that $v_1 \in C_1$, $V_2 \in C_2$ where $(v_1, v_2) \notin E$, and we know that $(v_1, d) \in E$, $(v_2, d) \in E$. The union of parents and non-parent neighbors of d is all of the neighbors of d, so all of the neighbors of d form a complete set. This contradicts that $(v_1, v_2) \notin E$, so d must be in exactly one maximal clique. d is connected to its parents so they are also in the maximal clique. □

References

Aji, S. M., & McEliece, J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, *46*, 325–343.

Allen, D., & Darwiche, A. (2003). New advances in inference by recursive conditioning. In *Proc. of the 19th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 2–10). San Francisco: Morgan Kaufmann.

Arnborg, S., Corneil, D. J., & Proskurowski, A. P. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, *8*, 227–284.

Bacchus, F., Dalmao, S., & Pitassi, T. (2003a). Algorithms and complexity results for #SAT and Bayesian inference. In *IEEE symposium on foundations of computer science (FOCS)* (pp. 340–351).

Bacchus, F., Dalmao, S., & Pitassi, T. (2003b). Value elimination: Bayesian inference via backtracking search. In *Proc. of the 19th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 20–28). San Francisco: Morgan Kaufmann.

Bartels, C. (2008). *Graphical models for large vocabulary speech recognition*. PhD thesis, University of Washington, Seattle.

Bartels, C., & Bilmes, J. (2004). *Elimination is not enough: Non-minimal triangulations for graphical models* (Tech. Rep. UWEETR-2004-0010). University of Washington.

Bartels, C., & Bilmes, J. (2006). Non-minimal triangulations for mixed stochastic/deterministic graphical models. In *Proc. of the 22nd conference on uncertainty in artificial intelligence*, Cambridge, MA, USA.

Bartels, C., Duh, K., Bilmes, J., Kirchhoff, K., & King, S. (2005). Genetic triangulation of graphical models for speech and language processing. In *Proc. of the 9th European conference on speech communication and technology (Eurospeech'05)*, Lisbon, Portugal.

Berre, D. L., & Simon, L. (2005). Preface to the special volume on the SAT 2005 competitions and evaluations. *Journal on Satisfiability, Boolean Modeling and Computation, 2*.

Bertele, U., & Brioschi, F. (1972). *Nonserial dynamic programming*. New York: Academic Press.

Bilmes, J., & Bartels, C. (2003). On triangulating dynamic graphical models. In *Proc. of the 19th conference on uncertainty in artificial intelligence*, Acapulco, Mexico.

Bilmes, J., Zweig, G. et al. (2001). Discriminatively structured dynamic graphical models for speech recognition. In *Final report: JHU 2001 summer workshop*.

Bodlaender, H. L., Koster, AM, van den Eijkhof, F., & van der Gaag, L. C. (2001). Pre-processing for triangulation of probabilistic networks. In *Proc. of the 17th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 32–39). San Francisco: Morgan Kaufmann.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proc. of the 12th conference on uncertainty in artificial intelligence*, Portland, Oregon.

Chavira, M., & Darwiche, A. (2007). Compiling Bayesian networks using variable elimination. In *Proc. of the 20th international joint conference on artificial intelligence (IJCAI)* (pp. 2443–2449).

Chavira, M., & Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence*, *172*(6–7), 772–799.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *STOC '71: Proc. of the third annual ACM symposium on theory of computing* (pp. 151–158). New York: ACM.

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, *126*, 5–41.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*(3), 201–215.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Artificial Intelligence*, *93*(1–2), 1–27.

Dechter, R. (1990). Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, *41*(3), 273–312.

Dechter, R. (1996). Bucket elimination: a unifying framework for several probabilistic inference algorithms. In *Proc. of the 12th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 211–219). San Francisco: Morgan Kaufmann.

Dechter, R. (1998). Bucket elimination: a unifying framework for probabilistic inference. In M.I. Jordan (Ed.), *Learning and inference in graphical models* (pp. 75–104). Cambridge: MIT Press.

Dechter, R. (2003). *Constraint processing*. San Francisco: Morgan Kaufmann.

Dechter, R., & Fattah, Y.E. (2001). Topological parameters for time-space tradeoff. *Artificial Intelligence*, *125*(1), 93–118.

Dechter, R., & Larkin, D. (2001). Hybrid processing of beliefs and constraints. In *Proc. of the 17th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 112–119). San Francisco: Morgan Kaufmann.

Dechter, R., & Mateescu, R. (2004). Mixtures of deterministic-probabilistic networks and their and/or search space. In *Proc. of the 20th annual conference on uncertainty in artificial intelligence (UAI)*, Arlington, Virginia (pp. 120–129).

Draper, D. (1995). Clustering without (thinking about) triangulation. In *Proc. of the 11th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 125–133).

Filali, K., & Bilmes, J. (2005). A dynamic Bayesian framework to model context and memory in edit distance learning: An application to pronunciation classification. In *Proc. of the 43rd annual meeting of the association for computational linguistics (ACL'05)* (pp. 338–345). Ann Arbor: Association for Computational Linguistics. http://www.aclweb.org/anthology/P/P05/P05-1042.

Filali, K., & Bilmes, JA (2007). Multi-dynamic Bayesian networks. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems 19* (pp. 409–416). Cambridge: MIT Press.

Fishelson, M., & Geiger, D. (2004). Optimizing exact genetic linkage computations. *Journal of Computational Biology*, *11*(2–3), 263–275.

Gavril, F. (1972). Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, *1*(2), 180–187.

Golumbic, M. C. (1980). *Algorithmic graph theory and perfect graphs*. New York: Academic Press.

Jensen, F., & Andersen, S. (1990). Approximations in Bayesian belief universes for knowledge-based systems. In *Proc. of the 6th annual conference on uncertainty in artificial intelligence (UAI)*, New York: Elsevier.

Jensen, F., & Jensen, F. (1994). Optimal junction trees. In *Proc. of the 10th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 360–366). San Francisco: Morgan Kaufmann.

Jensen, F., Lauritzen, S., & Olesen, K. (1990). Bayesian updating in causal probabilistic networks by local computation. *CSQ. Computational Statistics Quarterly*, *4*, 269–282.

Kjaerulff, U. (1990). *Triangulation of graphs-algorithms giving small total state space* (Tech. Rep. R-90-09). Aalborg University.

Kjærulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. of the 8th conference on uncertainty in artificial intelligence (UAI)* (pp. 121–129). San Francisco: Morgan Kaufmann.

Larkin, D., & Dechter, R. (2003). Bayesian inference in the presence of determinism. In *Proc. of the AISTATS, '03*. Key West: Soc. for AI and Statistics.

Larranaga, P., Kuijpers, C., Poza, M., & Murga, R. (1997). Decomposing Bayesian networks by genetic algorithms. *Statistics and Computing*, *7*(1), 19–34.

Lauritzen, S. (1996). *Graphical models*. London: Oxford University Press.

Lauritzen, S., & Spiegelhalter, D. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society*, *50*, 57–224.

Lei, X., Ji, G., Bilmes, J., & Ostendorf, M. (2005). DBN multistream models for Mandarin toneme recognition. In *Proc. of the international conference on acoustics, speech and signal processing (ICASSP)*.

Mateescu, R.E. (2007). *AND/OR search spaces for graphical models*. PhD thesis, University of California, Irvine.

Meila, M., & Jordan, M. I. (1997). Triangulation by continuous embedding. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems* (Vol. 9). Cambridge: MIT Press.

Murphy, K. P. (2002). *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley.

Nilsson, D. (1998). An efficient algorithm for finding the m most probable configurations in probabilistic expert systems. *Statistics and Computing*, *8*, 159–173.

Ohtsuki, T., Cheung, L. K., & Fujisawa, T. (1976). Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Mathematical Analysis and Applications*, *54*, 622–633.

Olesen, K. G., & Madsen, A. L. (2002). Maximal prime subgraph decomposition of Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics. Part B. Cybernetics*, *32*, 21–31.

Park, J. (2002). MAP complexity results and approximation methods. In *Proc. of the 18th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 388–396). citeseer.ist.psu.edu/park02map.html.

Parra, A., & Scheffler, P. (1995). How to use the minimal separators of a graph for its chordal triangulation. In *Automata, languages and programming* (pp. 123–134). citeseer.ist.psu.edu/parra94how.html.

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

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference* (2nd edn.). San Mateo: Morgan Kaufmann.

Rose, D. J. (1970). Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, *32*, 597–609.

Rose, D. J., Tarjan, RE, & Lueker, G. S. (1976). Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, *5*, 266–283.

Sang, T., Beame, P., & Kautz, H. (2005). Performing Bayesian inference by weighted model counting. In *Proc. of the 21st national conference on artificial intelligence (AAAI 05)*, Pittsburgh, USA.

Shenoy, P. P., & Shafer, G. (1986). Propagating belief functions with local computations. *IEEE Expert*, *1*(3), 43–52.

Subramanya, A., Gowdy, J., Bartels, C., & Bilmes, J. (2004). DBN based multi-stream models for audio-visual speech recognition. In *Proc. of the international conference on acoustics, speech and signal processing (ICASSP)*, Montreal, Canada.

Tarjan, R.E., & Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, *13*, 566–579.

Tsang, E. (1993). *Foundations of constraint satisfaction*. New York: Academic Press.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260–269.

Vomlel, J. (2002). Exploiting functional dependence in Bayesian network inference. In *Proc. of the 18th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 528–535). San Francisco: Morgan Kaufmann.

Wen, W. X. (1991). Optimal decomposition of belief networks. In *Proc. of the 6th annual conference on uncertainty in artificial intelligence (UAI)* (pp. 209–224). New York: Elsevier.

Xiang, Y. (1999). Temporally invariant junction tree for inference in dynamic Bayesian network. In *Lecture notes in computer science: Vol. 1600. Artificial intelligence today: recent trends and developments*. Berlin: Springer.

Yannakakis, M. (1981). Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, *2*(1), 77–79.

Zhang, L., & Malik, S. (2002). The quest for efficient boolean satisfiability solvers. In *Proc. of the 8th international conference on computer aided deduction (CADE 2002)*.

Zhang, Y., Diao, Q., Huang, S., Hu, W., Bartels, C., & Bilmes, J. (2003). DBN based multi-stream models for speech. In *Proc. of the IEEE intl. conference on acoustics, speech, and signal processing*, Hong Kong, China.

Zweig, G. G. (1998) *Speech recognition with dynamic Bayesian networks*. PhD thesis, University of California, Berkeley.