Cooperative Triangulation in MSBNs without Revealing Subnet Structures

Y. Xiang
Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2, yxiang@cs.uregina.ca

To appear in Networks

Abstract

Multiply sectioned Bayesian networks (MSBNs) provides a coherent framework for probabilistic inference in a cooperative multiagent distributed interpretation system. Inference in MSBNs can be performed effectively using a compiled representation. The compilation involves the triangulation of the collective dependency structure (a graph) defined in terms of the union of a set of local dependency structures (a set of graphs). Privacy of agents eliminates the option to assemble these graphs at a central location and to triangulate their union.

Earlier work solved distributed triangulation in a restricted case. The method is conceptually complex and the correctness of its extension to the general case is difficult to justify. In this paper, we present a new method that is conceptually simpler and is efficient. We prove its correctness in the general case and demonstrate its performance experimentally.

1 Introduction

Multiply sectioned Bayesian networks (MSBNs) provides a coherent framework for probabilistic inference in a large domain [9]. It can be applied under a single agent paradigm [8] or a cooperative multi-agent paradigm [6]. It supports object-oriented inference [3].

Inference in a MSBN can be performed effectively using a compiled representation called a linked junction forest (LJF) [9]. The compilation of a MSBN into a LJF takes several steps that are similar in principle to the compilation of a Bayesian network (BN) into its junction tree of belief universes [2]. This work focuses on the triangulation step. In particular, the input of the process is a set of undirected graphs with overlapping nodes which collectively defines a graph union G (to be defined formally), and the output is a corresponding set of chordal supergraphs which collectively defines a chordal supergraph of G.

Under the multiagent paradigm, each agent may be constructed by an independent vendor who embeds the know-hows about a subdomain into the agent. For example, the vendor of a component in a complex system can build an agent with the know-hows of the component embedded. The vendor may not be willing to reveal the know-hows when the agent is integrated into a multiagent MSBN. To protect the know-hows of such vendors, it is desirable not to force each agent to reveal its internal structure (the graph local to the agent). This privacy requirement eliminates the option to assemble these graphs at a central location and to triangulate their union as in the single-agent paradigm.

An algorithm to solve the problem under a restrictive condition was presented in [9]. The method is conceptually complex and the correctness of its extension to the general case is difficult to justify formally. In this work, we present a conceptually simple method to solve the problem and we prove its correctness in the general case.

In Section 2, we motivate this research with an overview of MSBNs and its applications. We briefly introduce the graph-theoretical terminologies to be used in the rest of the paper in Section 3. We define the problem of multiagent triangulation in Section 4. In Section 5 we present our method in the simplest case and prove its correctness. The method is extended to a more general case in Section 6. Building on the results of Sections 5 and 6, we present the most general case in Sections 7 and 8. The experimental study is presented in Section 9. We analyze the complexity of the proposed algorithms in Section 10.

2 Overview of MSBNs

In this section, we briefly introduce the framework of MSBNs. A BN [4] S is a triplet (N, D, P) where N is a set of domain variables, D is a DAG whose nodes are labeled by elements of N, and P is a joint probability distribution (jpd) over N specified in terms of probability distributions of each node in D conditioned on its parents.

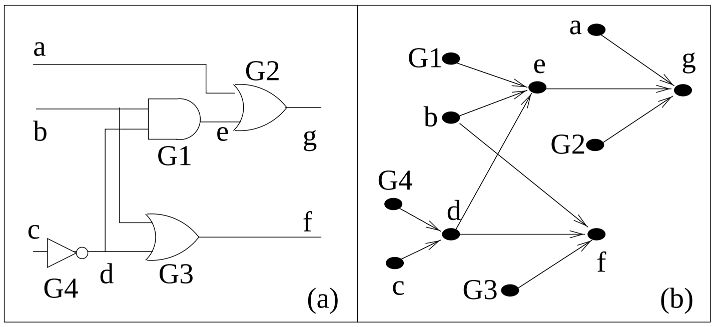

Figure 1: (a) A digital circuit. (b) The structure of a BN representing the circuit.

Figure 1 (a) shows a simple digital circuit. A BN (b) can be used to model the circuit for trouble-shooting purpose. The BN represents each device and each input/output as a node/variable, and represents a direct dependence between each pair of variables by an arc. The dependence is probabilistic in general, especially for the faulty behavior of devices. Each node is thus associated with a probability distribution conditioned on its parents. For example, the node g (output of an OR gate) is associated with the distribution $p(g|a, e, G_2)$. The distribution defines how the value of g depends on the values of its parent variables. For instance, $p(g = 0|a = 0, e = 0, G_2 = normal) = 1.0$ represents how input $a = 0$ and $e = 0$ to G_2 determine g when the gate is normal. On the other hand, $p(g = 0|a = 0, e = 0, G_2 = abnormal) = 0.3$ represents that g takes the correct value for the same input 30% of time when the gate is abnormal (and takes the wrong value 70% of the time). The BN can then be used to answer queries such as "$p(G_2 = abnormal | g = 1, a = 0, c = 1) =?$".

A MSBN [9] M is a collection of Bayesian subnets that together defines a BN. These subnets should satisfy certain conditions to permit coherent distributed inference. One condition requires that nodes shared by two subnets form a *d-sepset*, as defined below.

Let $G_i = (N_i, E_i)$ $(i = 0, 1)$ be two graphs. The graph $G = (N_0 \cup N_1, E_0 \cup E_1)$ is referred to as the *union* of G_0 and G_1, denoted by $G = G_0 \sqcup G_1$.

Definition 1 *Let $D_i = (N_i, E_i)$ $(i = 0, 1)$ be two DAGs such that $D = D_0 \sqcup D_1$ is a DAG. The intersection $I = N_0 \cap N_1$ is a* `d-sepset` *between D_0 and D_1 if for every $x \in I$ with its parents π in D, either $\pi \subseteq N_0$ or $\pi \subseteq N_1$. Each $x \in I$ is called a* `d-sepnode`.

Just as the structure of a BN is a DAG, the structure of a MSBN is a multiply sectioned DAG (MSDAG) with a hypertree organization:

Definition 2 *A* `hypertree MSDAG` *$\mathcal{D} = \bigsqcup_i D_i$, where each D_i is a connected DAG, is a connected DAG constructible by the following procedure:*

Start with an empty graph (no node). Recursively add a DAG D_k, called a `hypernode`, *to the existing MSDAG $\bigsqcup_{i=0}^{k\;1} D_i$ subject to the constraints:*
[d-sepset] For each D_j $(j < k)$, $I_{jk} = N_j \cap N_k$ is a d-sepset when the two DAGs are isolated.
[local covering] There exists D_i $(i < k)$ such that, for each D_j $(j < k; j \neq i)$, we have $I_{jk} \subseteq N_i$. For an arbitrarily chosen such D_i, I_{ik} is the `hyperlink` *between D_i and D_k which are said to be* `adjacent`.

We use $P_T(N)$ to denote a probability distribution over a set N of variables that is associated with an object T. A MSBN is then defined as follows:

Definition 3 *A MSBN M is a triplet $M = (\mathcal{N}, \mathcal{D}, \mathcal{P})$. $\mathcal{N} = \bigcup_i N_i$ is the* `total universe` *where each N_i is a set of variables. $\mathcal{D} = \bigsqcup_i D_i$ (a hypertree MSDAG) is the* `structure` *where nodes of each DAG D_i are labeled by elements of N_i. $\mathcal{P} = \prod_i P_{D_i}(N_i) / \prod_k P_{I_k}(I_k)$ is the* `jpd`. *Each $P_{D_i}(N_i)$ is a distribution over N_i such that whenever D_i and D_j are adjacent in $\mathcal{D}$, the marginalizations of $P_{D_i}(N_i)$ and $P_{D_j}(N_j)$ onto their d-sepset are identical. Each $P_{I_k}(I_k)$ is such a marginal distribution over a hyperlink I_k of $\mathcal{D}$. Each triplet $S_i = (N_i, D_i, P_i)$ is called a* `subnet` *of M. S_i and S_j are* `adjacent` *if D_i and D_j are adjacent.*

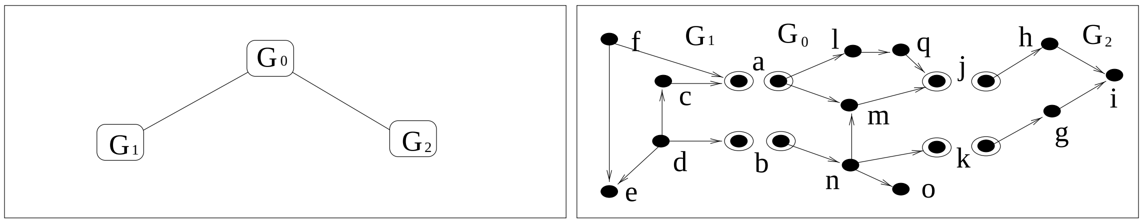

Figure 2: An example MSDAG of three DAGs.

Figure 2 illustrates MSBN with a trivial MSDAG. The hypertree is depicted in the left and the three DAGs are shown in the right. The d-sepnodes are highlighted with double ovals.

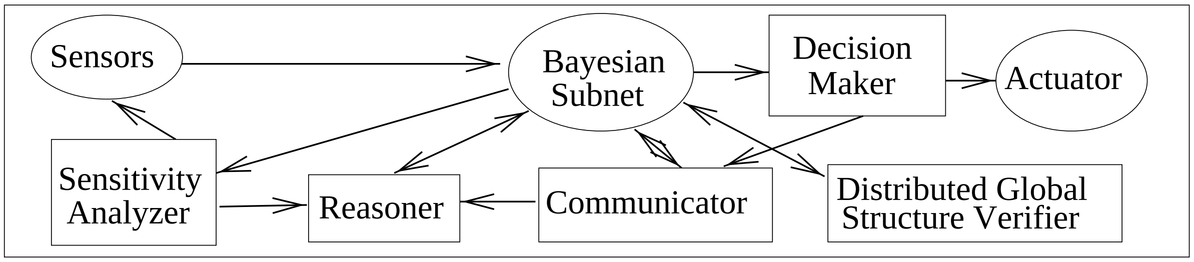

Figure 3: Main components of an agent in a MSBN-based multi-agent system.

MSBNs provide a framework for probabilistic reasoning in cooperative multi-agent distributed interpretation systems. Each agent holds its partial perspective of a large problem domain (Subnet in Figure 3), accesses a local evidence source (Sensors in Figure 3), communicates with other agents *infrequently* (Communicator), reasons with the local evidence and limited global evidence (Reasoner), and answers queries (Reasoner) or takes actions (Decision Maker/Actuator). If agents are cooperative, and each pair of adjacent agents are conditionally independent given their shared variables and have common initial belief on the shared variables, then a joint system belief is well defined which is identical to each agent's belief within its subdomain and supplemental to the agent's belief outside the subdomain [6]. Even though multiple agents may acquire evidence asynchronously in parallel, the communication operations of MSBNs ensure that the answers to queries from each agent are consistent with evidence acquired in the entire system after each communication. Since communication is infrequent, the operations also ensure that between two successive communications, the answers to queries for each agent are consistent with all local evidence gathered so far and are consistent with all evidence gathered in the entire system up to the last communication. Potential applications of the framework include decision support to cooperative human users in uncertain domains and troubleshooting a complex system by multiple knowledge based subsystems [6].

3 Graph-theoretical terminologies

In this section, we briefly introduce the graph-theoretic terminologies that the remaining of the paper depends on. Additional terminologies that are used but are not essential to understanding are included in the Appendix.

We shall call a graph $G = (N, E)$ as G over N. The *adjacency* of a node x is the set of nodes adjacent to x, and is denoted by $adj(x)$. A set X of nodes in G is *complete* if each pair of nodes in X is adjacent. A *chord* is a link connecting two nonadjacent nodes. G is *triangulated* or *chordal* if every cycle of length > 3 has a chord.

A node x in a graph $G = (N, E)$ is *eliminated* if $adj(x)$ is made *complete* by adding links (if necessary) before x and links incident to x are removed. Each link thus added is called a *fill-in*. Let F be the set of fill-ins added in eliminating all nodes in some order. Then the graph $G' = (N, E \cup F)$ is chordal. A graph is chordal iff all nodes can be eliminated one by one in some order without fill-ins [2].

Our method is based on node elimination in a certain order. We state this concept precisely:

Definition 4 *Let G be a graph over $N_1 \cup N_2$ such that $N_1 \cap N_2 = \phi$. G is eliminable in the order (N_1, N_2) if it is possible to eliminate all nodes in N_1 one by one first and then eliminate all nodes in N_2 one by one without any fill-ins.*

The concept can be extended such that the order is specified as a n-tuple. For simplicity, we shall write $(\{a\}, \{b\}, \{c\})$ as (a, b, c).

We define the concept of graph-consistency to refer to a pair of graphs whose subgraphs over shared nodes are identical.

Definition 5 *Let G_1 over N_1 and G_2 over N_2 be two graphs such that $N_1 \cap N_2 \neq \phi$. Then G_1 and G_2 are said to be* `graph-consistent` *if the subgraphs of G_1 and G_2 spanned by $N_1 \cap N_2$ are identical.*

4 The problem of cooperative triangulation

Performing inference directly in a MSBN is difficult since its MSDAG is often multiply connected. As one of the most efficient ways of inference in a multiply connected BN uses a junction tree (JT) (Appendix) representation, we compile a MSBN into a set of inter-related JTs called a linked junction forest (LJF) [9] for inference. In the LJF, the JTs are organized into the identical hypertree structure as the MSDAG. The compilation of a MSBN into a LJF takes several steps, some of which parallel the compilation of a BN. We discuss only the first four steps that are relevant to the purpose of this paper:

The first step, called *moralization*, is to convert the MSDAG into its *moral* graph by completing parents for each node and dropping the directions of links. Since the MSDAG is the union of a set of DAGs, its moral graph is defined in terms of the union of moral graphs of these DAGs.

The second step is to triangulate the moral graph of the MSDAG into a chordal graph. This is needed for the same reason in compiling a BN: a JT of a graph exists iff the graph is chordal. Again, this chordal graph is defined as the union of a set of chordal graphs each of which is obtained from the moral graph of a DAG. Each component graph should be chordal since we want to organize each subnet into a JT for local inference.

The third step is to convert each chordal graph in the union into a junction tree (JT) of belief universes [2] to be used in the inference computation.

The fourth step is to compile each hyperlink (d-sepset) of MSDAG into a *linkage tree* [6]. This step allows an agent's belief on d-sepset (a probability distribution) to be factorized compactly so that communication between agents can be performed more efficiently when the d-sepset is large. More details on this can be found in [5]. We give here a definition of linkage trees equivalent to that in [6] (but computationally less efficient) to facilitate the discussion. The proof of equivalence is trivial.

Definition 6 *Let I be the d-sepset between JTs T_a and T_b in a LJF. Repeat the following until no variable can be removed:*
(1) Remove a variable $x \notin I$ if x is contained in a single clique C.
(2) If C becomes a subset of an adjacent clique D after (1), union C into D.

Let L be the graph resultant from the procedure. Then L is a `linkage tree` *of T_a with respect to I if $\cup_{l \in L} l = I$, where each clique l in L is a* `linkage`. *Define a clique in T_a that contains l as its* `linkage host` *and break ties arbitrarily.*

It can be shown [7] that the linkage tree L is a JT, and that if T_a is an I-map (Appendix), then L is an I-map over I. Therefore, to propagate belief on I from T_a to T_b during inference, it is sufficient to propagate belief on each linkage [5] since

$$B(I) = [\prod_l B(l)]/[\prod_q B(q)],$$

where each l is a linkage in L with its belief table $B(l)$ and each q is a separator in L with its belief table $B(q)$.

On the other hand, if $\cup_{l \in L} l \supset I$ when the procedure in Definition 6 halts, in which case L is *not* a linkage tree, then $B(I)$ cannot be compactly represented by the above equation and can only be obtained by more expensive computation [9]. The following theorem identifies the condition under which the linkage tree exists[1]. The condition is that the graph from which T_a is constructed must be triangulated in a certain way.

Theorem 7 *Let G be a graph over N from which a JT T is constructed. Let I be a subset of nodes in G.*

Then a linkage tree of T exists with respect to I iff G is eliminable in the order $(N \setminus I, I)$.

Proof:

[Sufficiency] Suppose G is eliminable in $(N \setminus I, I)$. Consider a node $x \in N \setminus I$ that can be eliminated first without fill-ins. Then x must appear in a single clique C in T since otherwise $adj(x)$ is incomplete. Hence x can be removed from C.

Repeating this argument for each node in $N \setminus I$, we will be able to eliminate $N \setminus I$ from T and the resultant graph is the linkage tree.

[Necessity] Suppose G is not eliminable in $(N \setminus I, I)$. That is, $N \setminus I$ cannot be eliminated without fill-ins in any particular order that is consistent with $(N \setminus I, I)$. This means that no matter what order we use, there exists a non-empty subset of $N \setminus I$ (the subset may differ for different orders) such that each node in the subset appears in at least two cliques of T. Hence the linkage tree does not exist. $\square$

For each JT T in the hypertree, belief propagation needs to be performed relative to each adjacent JT. To ensure the existence of linkage tree for each adjacent JT, we have the following requirement for triangulation:

Requirement 1 *For each hypernode in the hypertree of a MSBN, the triangulated moral graph G over N for the hypernode must be such that for every hyperlink I incident to the hypernode, G is eliminable in the order $(N \setminus I, I)$.*

For each hyperlink in the hypertree, the two endpoints (hypernodes) may potentially connect the d-sepset differently during triangulation. The following example shows that such triangulation should be prevented.

[1]This condition can be shown to be equivalent to the *host composition* condition in [9]. However, the host composition condition is *descriptive* while the condition presented here is *procedural* and hence provides direct guideline to triangulation.

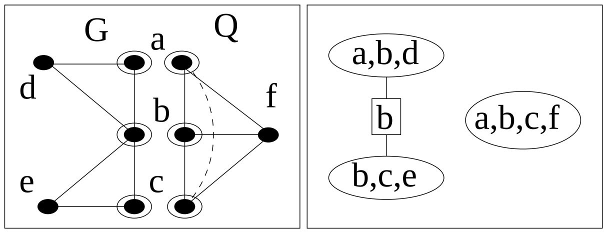

Figure 4: Illustration of Requirement 2. The d-sepset is shown by double ovals.

Consider the two moral graphs G and Q (ignore the dashed link) in Figure 4 (left). Using the order in Theorem 7, G is eliminated without any fill-ins and Q is eliminated with the fill-in $\{a, c\}$ (dashed). The two corresponding JTs are shown in the right. Note however, that the JT from G expresses a *false* conditional independence relation, i.e., that $\{a, d\}$ is independent of $\{c, e\}$ given b. To avoid this problem, the d-sepset $I = \{a, b, c\}$ should be *identically* connected in G and Q (by adding the fill-in $\{a, c\}$ to G). Hence, we have the following requirement for triangulation:

Requirement 2 *For each pair of adjacent hypernodes in the hypertree of a MSBN, the corresponding d-sepset must be connected identically in the two triangulated moral graphs. That is, the two triangulated graphs must be graph-consistent.*

The semantics of Requirement 2 is that assumptions on the conditional independence within a d-sepset expressed in different JTs must be consistent.

Under the multiagent paradigm, a MSBN may be integrated from a set of agents, say, for trouble-shooting a complex system. The MSBN designer may know only the interface (d-sepset) between subdomains (subnets internal to agents) but not the details of each subnet. Each agent may be constructed by an independent vendor who embeds the know-hows about a subdomain into the agent. For example, the vendor of a component in a complex system can build an agent with the know-hows of the component embedded. The vendor may not be willing to reveal the know-hows when the agent is integrated into a multiagent MSBN. To protect the know-hows of such vendors, it is desirable not to force each agent to reveal its internal structure during compilation. An agent will be given, by the MSBN designer, the information which other agents it interfaces to and what are the d-sepsets, but not much more than that regarding the internal of other agents. This privacy requirement eliminates the option to assemble DAGs in the MSDAG at a central location for compilation. In other words, it is desirable to distribute all steps of compilation.

The first step (moralization) can be distributed easily. First, each agent performs moralization locally. Then adjacent agents in the hypertree exchange links among d-sepnodes so that each pair of d-sepnodes are connected identically at different local graphs. Hence privacy of agents is not compromised in this step. Once the triangulation (second step) is completed, the third and fourth steps (construction of JT from the chordal graph and construction of the linkage tree from JT) are local and they do not threaten privacy of agents. We state the privacy requirement for the second step as follows:

Requirement 3 *For each hypernode H in the hypertree of a MSBN and each of its adjacent hypernode H', the internal dependency structure of H beyond the subgraph spanned by their d-sepset should not be revealed to H'.*

The focus of this work is the distribution of the second step: triangulation. The problem can be abstracted as follows: A set of agents is organized into a hypertree where each hypernode is labeled by an agent. Each agent has an undirected graph embedded in it. Each pair of graphs embedded in adjacent agents on the hypertree share a set of common nodes and are graph-consistent[2]. We shall refer to agents and graphs interchangeably. The graphs are so located on the hypertree that whenever two graphs have a set S of shared nodes with a third graph on the hyperpath between them, S is also shared by the third graph[3]. The problem is to find an effective way in which each agent triangulates its local graph subject to Requirements 1 through 3 such that the union of all local chordal graphs is triangulated. We shall refer to this problem as the problem of *cooperative triangulation*.

An algorithm to solve the problem in a hyperstar (a degenerated hypertree) was presented in [9]. The method is conceptually complex (e.g., six types of links and a concept d-path are defined, and the search of such d-paths is necessary) and its correctness in the general hypertree case is difficult to justify formally. In this paper, we present a conceptually simple method based on node elimination [1] and we prove its correctness in the general case.

We present the method stepwise and address only Requirements 2 and 3 in Sections 5 through 7. Requirement 1 is addressed in Section 8.

5 Cooperative triangulation of two agents

In this section, we consider the simplest case of the problem of cooperative triangulation, where only two agents/graphs are involved. Algorithm 1 does the cooperative triangulation. Given a set F of links over a set N of nodes, we shall call a subset $E \subseteq F$ a *restriction* of F to $S \subset N$ if

$$E = \{(x,y) | x \in S,\ y \in S,\ (x,y) \in F\}.$$

Algorithm 1

Description: Let N, V and S be disjoint nonempty sets of nodes. Let G be a graph over $N \cup S$ and Q be a graph over $V \cup S$ such that G and Q are graph-consistent. Let A and B be two agents such that G is embedded in A and Q is embedded in B.

```
begin
    1 A eliminates nodes in $G$ in the order $(N, S)$; denote the fill-ins by $F$;
    2 A adds $F$ to $G$; denote the resultant graph by $G'$;
    3 A sends B the restriction of $F$ to $S$;
    4 B adds to $Q$ the restriction of $F$ to $S$ received; denote the resultant graph by $Q'$;
    5 B eliminates nodes in $Q'$ in the order $(V, S)$; denote the fill-ins by $F'$;
    6 B adds $F'$ to $Q'$; denote the resultant graph by $Q''$;
    7 A sends A the restriction of $F'$ to $S$;
    8 B adds to $G'$ the restriction of $F'$ to $S$ received; denote the resultant graph by $G''$;
end
```

[2]This condition can be ensured by distributed moralization.

[3]This is equivalent to the local covering condition.

Algorithm 1 clearly satisfies the privacy requirement since each exchange between the two agents reveals only the connection over S. Figure 5 illustrates the algorithm, where graphs G and Q are shown in the left. The shared nodes $S = \{a, b\}$ are shown as double ovals. Elimination in G is performed in the order $(\{e, f, c, d\}, \{a, b\})$. Node e can be eliminated without fill-in. No nodes in $\{f, c, d\}$ can now be eliminated without fill-ins. If f is eliminated next, then a fill-in $\{a, d\}$ is required. Node c can then be eliminated without fill-in. To eliminate d next, another fill-in $\{a, b\}$ is needed. Hence elimination in G in the order (e, f, c, d, a, b) produces G' in Figure 5 (right), where the fill-ins are shown as dashed lines.

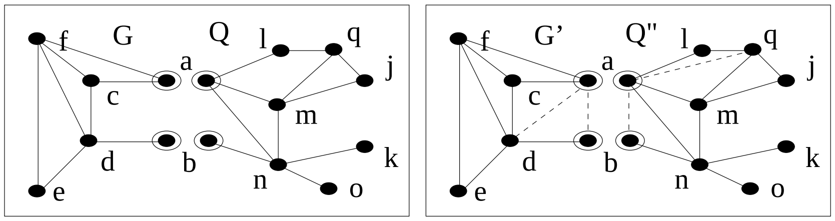

Figure 5: Illustration of Algorithm 1.

The fill-in $\{a, b\}$ is then added to Q to obtain Q' (not shown). Elimination in Q' in the order $(k, o, j, l, q, m, n, a, b)$ produces fill-in $\{a, q\}$ and the resultant graph Q'' is shown in Figure 5 (right). For this example, no fill-ins from Q'' need to be added to G', and hence $G'' = G'$. However, this is not always the case.

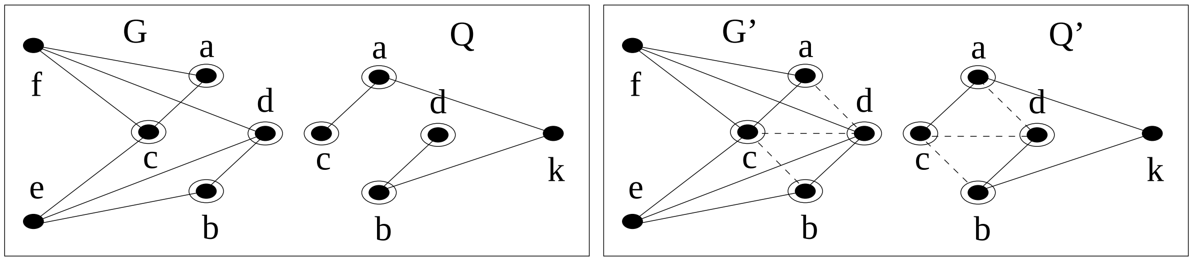

Figure 6: Illustration of last two steps of Algorithm 1.

To see the need of steps 7 and 8 in Algorithm 1, consider the example in Figure 6. The graphs G and Q are shown in the left, where $S = \{a, b, c, d\}$. After elimination in G in the order (e, f, a, b, c, d), the resultant G' is shown in Figure 6 (right). After the restriction of fill-ins to S is added to Q, we obtain Q' in Figure 6 (right). Elimination in Q' should start with k, which requires a fill-in $\{a, b\}$. This is the only fill-in added to obtain Q'' (not shown). Since $\{a, b\}$ is between nodes in S, it will be added to G' to obtain G'' (not shown) in steps 7 and 8 of Algorithm 1.

The following proposition shows that the resultant graphs from Algorithm 1 are graph-consistent and their union is chordal:

Proposition 8 *When Algorithm 1 halts, the following hold:*
(1) G'' and Q'' are graph-consistent.
(2) The graph union $G'' \sqcup Q''$ is chordal.

Proof:
(1) It is true from steps 4 and 8.
(2) It suffices to show that nodes in $G'' \sqcup Q''$ is eliminable in the order (V, N, S).
V can be eliminated first without fill-ins since the subgraph of $G'' \sqcup Q''$ spanned by $V \cup S$ is Q'' by steps 7 and 8, and Q'' is eliminable in the order (V, S) from steps 5 and 6. The remaining graph is G'' by step 8.
N can be eliminated first from G'' without fill-ins since subgraphs of G'' and G' spanned by N are identical by step 8, and G' is eliminable in the order (N, S) by steps 1 and 2. The remaining graph is spanned by S. It is eliminable since Q'' is eliminable in the order (V, S). □

Proposition 8 not only serves to illustrate the basic idea of our method, it is also needed in proving the more general case below.

6 Cooperation with hyperstar organization

Next, we consider cooperative triangulation where $n > 2$ agents are organized into a hyperstar. We denote the agent at the center of the hyperstar by A_0, and the agent at each leaf by A_i $(i = 1, ..., n \quad 1)$. We denote the graph embedded in agent A_i $(i = 0, ..., n \quad 1)$ by G_i over nodes N_i. We assume that these graphs satisfy the local covering condition: $N_i \cap N_j \subset N_0$ for every i and j $(i \neq j)$. We denote $N_0 \cap N_i$ by S_i.

Algorithm 2

```
set $LINK = \phi$;
for each leaf agent $A_i$, do
    eliminate $N_0$ in the order $(N_0 \setminus S_i, S_i)$ and denote the resultant fill-ins by $F$;
    add $F$ to $G_0$ and $LINK$;
    send $A_i$ the restriction of $F$ to $S_i$;
    receive a set $F'$ of fill-ins over $S_i$ from $A_i$;
    add $F'$ to $G_0$ and $LINK$;
denote the resultant graph by $G'_0$;

for each leaf agent $A_i$, do
    send $A_i$ the restriction of $LINK$ to $S_i$;
```

Algorithm 2 is executed by A_0. It is organized into two stages indicated by a blink line in-between.

Algorithm 3 is executed by each A_i $(i = 1, ..., n \quad 1)$. It is also organized into two stages. We assume that if an existing link is to be added to a graph, it has no effect.

Algorithm 3

```
receive a set $F$ of fill-ins over $S_i$ from $A_0$;
add $F$ to $G_i$;
eliminate $N_i$ in the order $(N_i \setminus S_i, S_i)$ and denote the resultant fill-ins by $F'$;
add $F'$ to $G_i$;
send $A_0$ the restriction of $F'$ to $S_i$;

receive a set $LINK'$ of fill-ins over $S_i$ from $A_0$;
add $LINK'$ to $G_i$ and denote the resultant graph by $G_i'$;
```

The process starts by A_0. In each iteration of the first *for* loop (Algorithm 2), it performs a local elimination relative to a S_i, adds fill-ins locally and sends them to A_i. When A_i receives the fill-ins (Algorithm 3), it updates its graph, performs a local elimination relative to S_i, adds fill-ins locally and sends them back to A_0. After the above has been done with each A_i, A_0 finalizes G_0' and starts the second *for* loop (Algorithm 2). It sends all relevant fill-ins obtained in the first loop to each A_i. In response, each A_i receives the fill-ins and finalizes G_i' (Algorithm 3).

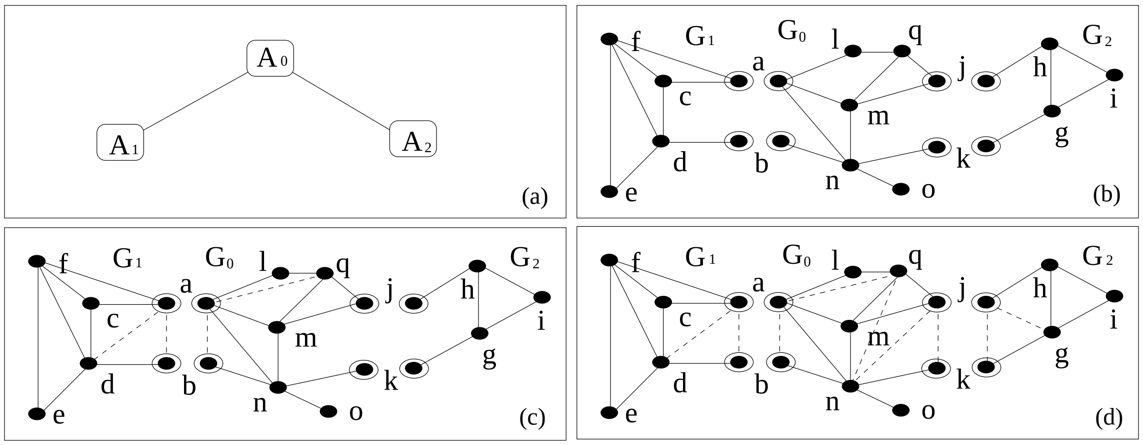

Figure 7: Illustration of Algorithm 2 and 3.

Figure 7 illustrates Algorithms 2 and 3. The hyperstar with three agents is shown in (a), and the three graphs are shown in (b). After A_0 iterated once in the first *for* loop (relative to A_1) and A_1 completed its first stage, the resultant graphs are shown in (c). The corresponding elimination orders are $(o, k, j, l, q, m, n, a, b)$ and (e, f, c, d, a, b), respectively. After A_0 iterated the second time in the first *for* loop (relative to A_2) and A_2 completed its first stage, the resultant graphs are shown in (d). The corresponding elimination orders are $(o, l, b, a, q, m, n, j, k)$ and (i, h, g, j, k), respectively. For this example, no fill-ins are to be distributed in the second stage of Algorithms 2 for A_0. This is not the case in the next example.

Suppose the three agents on the hyperstar of Figure 7(a) are associated with the graphs in Figure 8(a). After the first iteration of A_0 in the first *for* loop and the first stage of A_1, the resultant graphs are shown in (b). The elimination orders are (k, d, c, b, a) and (i, d, b, a),

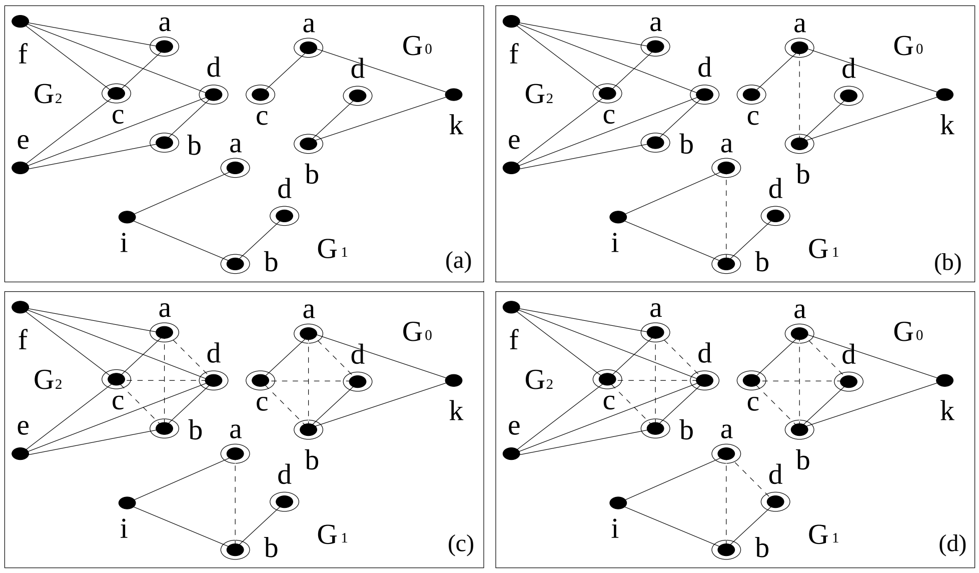

Figure 8: Illustration of Algorithm 2 and 3.

respectively. After the second iteration of A_0 in the first *for* loop and the first stage of A_2, the resultant graphs are shown in (c). The elimination orders are (k, d, c, b, a) and (f, e, d, c, b, a), respectively. After the second stage of A_0, the resultant graphs are shown in (d). Note that without this stage, the fill-in $\{a, d\}$ cannot be added to G_1.

Also note that in the second stage of A_0, it is unnecessary to iterate through the leaf agent involved in the last iteration of the first *for* loop (i.e., A_2 in this example). We did not exclude this iteration in Algorithm 2 in order to keep it simple.

Clearly, Algorithms 2 and 3 satisfy the privacy requirement. We show that the resultant graphs is graph-consistent and their union is chordal:

Proposition 9 *When Algorithms 2 and 3 halt at all corresponding agents, the following hold:*
(1) For $i = 1, ..., n \quad 1$, G'_0 *and* G'_i *are graph-consistent.*
(2) The graph union $\sqcup_{i=0}^{n \quad 1} G'_i$ *is chordal.*

Proof:

(1) For each i, G_0 and G_i are graph-consistent by assumption (Section 4). All fill-ins over S_i during the entire process are added to G'_0, accumulated in $LINK$, and sent to A_i by A_0. They are added to G'_i by A_i. Hence, G'_0 and G'_i are graph-consistent.

(2) We prove the result through a modified version of Algorithms 2 and 3:

Without losing generality, we shall assume that the first *for* loop in Algorithm 2 proceeds in the order $i = 1,\ 2, ...,\ n \quad 1$. For Algorithm 2, we add to the modified version the following as the last step of the first *for* loop:

```
send $A_j$ ($j = 1, ..., i \quad 1$) the restriction of $LINK$ to $S_j$;
```

For Algorithm 3, we embed its second stage (the last two lines) into a loop that iterates i times, where the first $i \quad 1$ iterations corresponds to the additional sending by A_0.

The effect of the modification is that as soon as A_0 and A_i have each completed local elimination relative to each other, the new fill-ins are immediately communicated to A_1 through $A_{i\ \ 1}$ and are included in their local graphs. Therefore, the modified version will produce exactly the same end result as the original algorithms.

Let G_0^1 denote the modified G_0 after A_0 has completed the first iteration of the first *for* loop ($i = 1$). We shall use the value of i as the superscript to index the current graphs in different agents. Let G_1^1 denote the modified G_1 after A_1 has completed its first stage. According to Proposition 8, the graph union $Q_1^1 = G_0^1 \sqcup G_1^1$ is chordal.

After A_0 has completed the second iteration of the first *for* loop ($i = 2$), it updates G_0^1 into G_0^2. According to the modified algorithm, A_0 will communicate the new fill-ins over S_1 to A_1 so that A_1 can update G_1^1 into G_1^2. This updating by A_0 and A_1 can be collectively considered as equivalent to updating Q_1^1 into $Q_1^2 = G_0^2 \sqcup G_1^2$. Note that G_0^2 and G_1^2 are graph-consistent. The updating can also be considered as an elimination by A_1 followed by an elimination by A_0 as described in Algorithm 1. Hence Q_1^2 is chordal according to Proposition 8.

On the A_2 side, after A_2 has completed its first stage, it updates G_2 into G_2^2. The updating by A_0, A_1 and A_2 can be collectively considered as equivalent to an elimination on Q_1^1 followed by an elimination on G_2 as described in Algorithm 1 because the local covering condition holds. Applying Proposition 8, we conclude that the graph union $Q_2^2 = Q_1^2 \sqcup G_2^2 = G_0^2 \sqcup G_1^2 \sqcup G_2^2$ is chordal.

Using the above argument recursively on i ($i = 3, ..., n \quad 1$), the end result $Q_{n\ \ 1}^{n\ \ 1} = G_0^{n\ \ 1} \sqcup ... \sqcup G_{n\ \ 1}^{n\ \ 1} = \sqcup_{i=0}^{n\ \ 1} G'_i$ is chordal. □

Note the use of the local covering condition in the above proof. The chordality result would not hold without the local covering condition.

Note also that just as the additional communication in the modified version of Algorithms 2 and 3 used in the proof does not affect the end result, the statement "add F' to G_i" in Algorithm 3 does not affect the end result and can be removed.

7 Cooperation with hypertree organization

We now consider the most general case of the cooperative triangulation problem, where $n > 3$ agents are organized into a hypertree.

We present recursive algorithms for each agent. The execution of each algorithm by an agent is activated by a call from an entity known as *caller*. We denote the agent called to execute the algorithm by A_0. The caller is either an adjacent agent of A_0 in the hypertree or the system. If caller is an agent, we denote it by A_c (with graph G_c over N_c embedded). If A_0 has adjacent agents other than A_c, we denote them by $A_1, ..., A_n$. We denote $N_c \cap N_0$ by S_c and $N_0 \cap N_i$ by S_i ($i = 1, ..., n$).

In algorithm DepthFirstEliminate, an agent A_0 performs elimination and updating with respect to all adjacent agents.

In algorithm DistributeDlink, an agent A_0 receives fill-ins from caller and then distributes all fill-ins among d-sepnodes added to G_0 since the start of the cooperative triangulation to each other adjacent agent.

Algorithm CoTriangulate is executed by the system to activate the cooperative triangulation by multiple agents.

Algorithm 4 (DepthFirstEliminate)

```
if caller is an agent $A_c$, do
    receive a set $F_c$ of fill-ins over $S_c$ from $A_c$;
    add $F_c$ to $G_0$;

set $LINK = \phi$;
for each agent $A_i$ $(i = 1, ..., n)$, do
    eliminate $N_0$ in the order $(N_0 \setminus S_i, S_i)$ and denote the resultant fill-ins by $F$;
    add $F$ to $G_0$ and $LINK$;
    send $A_i$ the restriction of $F$ to $S_i$;
    call $A_i$ to run DepthFirstEliminate and receive fill-ins $F'$ over $S_i$ from $A_i$ when finished;
    add $F'$ to $G_0$ and $LINK$;

if caller is an agent $A_c$, do
    eliminate $N_0$ in the order $(N_0 \setminus S_c, S_c)$ and denote the resultant fill-ins by $F'_c$;
    add $F'_c$ to $G_0$ and $LINK$;
    send $A_c$ the restriction of $LINK$ to $S_c$;
```

Algorithm 5 (DistributeDlink)

```
if caller is an agent $A_c$, do
    receive a set $F_c$ of fill-ins over $S_c$ from $A_c$;
    add $F_c$ to $G_0$;

set $LINK$ to the set of all fill-ins added to $G_0$ so far;
for each agent $A_i$ $(i = 1, ..., n)$, do
    send $A_i$ the restriction of $LINK$ to $S_i$;
```

Algorithm 6 (CoTriangulate)

```
choose an agent $A_*$ arbitrarily;
call $A_*$ to run DepthFirstEliminate;
after $A_*$ has finished, call $A_*$ to run DistributeDlink;
```

Figure 9 illustrates CoTriangulate with a system of 11 agents. The hypertree is depicted in the figure with each node labeled by an agent. Suppose the agent A_* chosen in the algorithm is A_5. For each arrow, an elimination (see DepthFirstEliminate) by the agent at the tail on its local graph is performed and the relevant fill-ins generated are then sent to the agent at the head. For example, the arrow from A_5 to A_3 represents the elimination of A_5 on G_5 in the order $(N_5 \setminus N_3, N_5 \cap N_3)$, followed by sending A_5 the restriction of fill-ins to $N_5 \cap N_3$. The label of each arrow shows the order of the operation. It's easy to see that the order is similar to a depth-first traversal and hence the name of the operation.

After A_5 has finished DepthFirstEliminate, the flow of fill-ins during execution of DistributeDlink is shown by *only* those arrows pointing away from A_5.

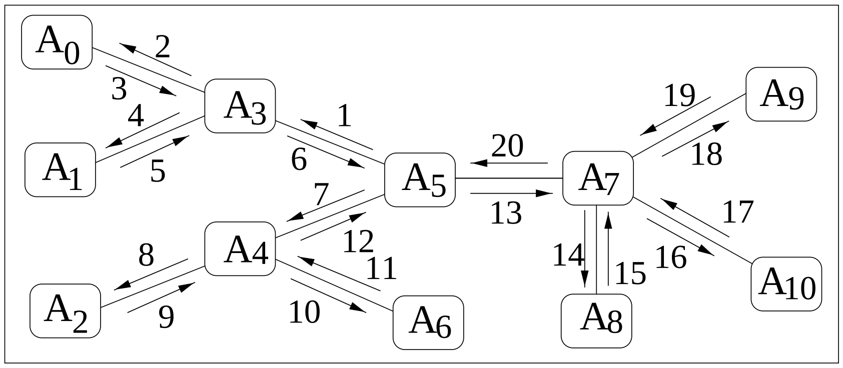

Figure 9: Illustration of CoTriangulate.

In Theorem 11, we show the properties of CoTriangulate. We define the *altitude* of a node in a hypertree to be used in the proof. Given a node A_0 on the hypertree, find the longest path from A_* through A_0 to a leaf, and denote the leaf by A_f. The altitude of A_0 is then the length of the path between A_0 and A_f. In Figure 9, for example, if A_* is A_5, then the altitudes of A_2, A_4 and A_5 are 0, 1 and 2, respectively.

Lemma 10 establishes the depth-first property of DepthFirstEliminate to be used in proving Theorem 11. We shall regard A_* as the *root* of the hypertree.

Lemma 10 *All eliminations on graphs located at the sub(hyper)tree rooted at A_0 are performed after A_0 is called to run DepthFirstEliminate and before A_0 returns from the call.*

Proof:

We prove by induction on the altitude k of A_0.

When $k = 0$, A_0 is a leaf and exactly one elimination is performed on G_0 (see Figure 9, where the total number of eliminations on a graph located at a (hyper)node is shown by the number of outgoing arrows of the node). Only the two *if* statements in DepthFirstEliminate are executed in this case, where the second one contains the elimination. Hence the lemma is true.

Assume that the lemma is true when $k = m \geq 0$. Now consider the case $k = m + 1$. The eliminations performed on graphs located at the subtree rooted at A_0 are those performed on G_0 and those performed on graphs located at the subtree rooted at each A_i ($i = 1, ..., n$). Exactly $n + 1$ eliminations are to be performed on G_0 (one relative to each A_i ($i = 1, ..., n$) and one relative to A_c). The n eliminations relative to A_i ($i = 1, ..., n$) are contained in the *for* loop. All eliminations on graphs located at the subtree rooted at each A_i ($i = 1, ..., n$) are also performed in the *for* loop during the call to A_i by the inductive assumption. The last elimination on G_0 is performed in the *if* statement following the *for* loop. $\square$

We now prove the properties of CoTriangulate.

Theorem 11 *When CoTriangulate halts, the following hold:*
(1) Each pair of adjacent graphs on the hypertree are graph-consistent.
(2) The graph union of all graphs on the hypertree is chordal.

Proof:

(1) This is true due to the execution of DistributeDlink by agent A_*, which recursively propagates fill-ins outwards from A_* to the entire hypertree.

(2) We prove using a modified DepthFirstEliminate where the following statement is added at the end:

```
for each agent $A_j$ ($j = 1, ..., n$), do
    send $A_j$ the restriction of $LINK$ to $S_j$ and call $A_j$ to run DistributeDlink;
```

Its effect is that as soon as all eliminations on graphs located in the subtree rooted at A_0 has completed (Lemma 10), adjacent graphs on the subtree are made graph-consistent. Hence, the end result of CoTriangulate using the modified DepthFirstEliminate is invariant.

Using the modified algorithm, we prove by induction on the altitude k of agent A_0. When $k = 1$, A_1 through A_n are leaves of the hypertree. By Lemma 10, all eliminations on graphs located on the subtree rooted at A_0 are performed when A_0 is called to run DepthFirstEliminate. The processing is identical to that of Algorithms 2 and 3, except the additional elimination in G_0 relative to A_c. Hence by Proposition 9, the graph union Q of G_0 through G_n is chordal before the additional elimination. Since the additional elimination and the updating that follows change neither the chordality of G_0 nor that of the remaining subgraph of Q, the graph union of G_0 through G_n is chordal after the additional elimination.

Now assume that when the altitude of A_0 is $k = m \geq 1$, the graph union of G_0 through G_n is chordal after A_0 is called to run the modified DepthFirstEliminate.

Consider the case $k = m + 1$. When DepthFirstEliminate is called by A_0 in each A_i ($i = 1, ..., n$), it is equivalent to regard A_i as a (hyper)leaf with Q_i embedded in it, where Q_i is the union of all graphs on the subtree rooted at A_i. It is equivalent since all eliminations on graphs in the subtree rooted at A_i are performed during the call of DepthFirstEliminate on A_i by Lemma 10, and since the local covering condition holds. By the inductive assumption, when the call is completed, Q_i is chordal. Due to the equivalence with the processing of Algorithms 2 and 3, we conclude that the graph union of G_0, $Q_1, ..., Q_n$ is chordal by Proposition 9. $\square$

Theorem 11 shows that when CoTriangulate halts, the union of all graphs on the hypertree is chordal. Recall that each local graph needs to be chordal as well (Section 4). Theorem 12 shows that this is automatically satisfied due to the way in which a MSDAG is defined.

Theorem 12 *Let G be a chordal graph and g be a graph obtained by deleting some nodes (and links incident to these nodes) from G. Then g is chordal.*

Proof:

It suffices to show that after the deletion of a single node x, the remaining graph g is chordal. We prove by contradiction:

Suppose that g is nonchordal. Then there must be a chordless cycle c of length > 3 in g. Since G is chordal, c should be produced by the deletion of x and links incident to x. However, no matter what the adjacency of x is in G, we cannot make the chordless cycle c to go away. Hence, c is in G before x is deleted. This implies that G is nonchordal: a contradiction. $\square$

8 Ensure elimination orders for multiple d-sepsets

In Section 7, we have presented CoTriangulate that solves the cooperative triangulation problem subject to Requirements 2 and 3. Would Requirement 1 be satisfied as well? The following example shows that this is not the case in general.

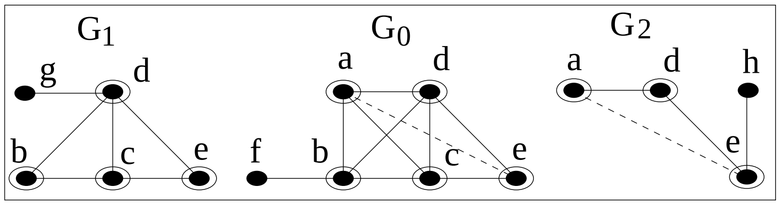

Figure 10: Illustration of violation of Requirement 1.

Consider the three local graphs (ignore the dashed links) in Figure 10 with the hyperstar G_1 G_0 $G2$. Following CoTriangulate, agent A_0 first eliminates G_0 in the order (f, a, b, c, d, e) relative to A_1. The elimination produces no fill-ins. Then agent A_1 eliminates G_1 in the order (g, b, c, d, e) relative to A_0 also without fill-ins.

Next, agent A_0 eliminates G_0 relative to A_2 in the order (f, b, c, a, d, e), which produces fill-in $\{a, e\}$ (the dashed line in G_0). It will be sent to A_2 and added to G_2 (the dashed line in G_2). A_2 then eliminates G_2 in the order (h, a, d, e) relative to A_0 without additional fill-ins. CoTriangulate now halts.

However, if we eliminate the new G_0 again relative to A_1 in the order (f, a, b, c, d, e), another fill-in $\{b, e\}$ is now needed. Hence, CoTriangulate does not guarantee satisfaction of Requirement 1 in general. On the other hand, situations like the above example do not seem to arise often. In our experimental study (see Section 9) with three MSBNs, all of them satisfied Requirement 1 when CoTriangulate halted. Hence we suggest the following algorithm to address Requirement 1:

Algorithm 7 (SafeCoTriangulate)

```
perform CoTriangulate;
each agent performs an elimination relative to the d-sepset with each adjacent agent;
if no agent added any fill-ins, halt;
else restart this algorithm;
```

Note that the elimination performed in SafeCoTriangulate is strictly *local* to each agent. See Section 10 for the complexity analysis and possible measures for improving the efficiency. We have the following theorem on the property of SafeCoTriangulate:

Theorem 13 *SafeCoTriangulate will halt, and when it halts, Requirement 1 is satisfied.*

Proof:

Each round of SafeCoTriangulate will add some fill-ins to some local graphs, since otherwise it will halt. Since only a finite number of fill-ins can be added to each local graph before it becomes complete, and a complete graph is eliminable in any order, we conclude that SafeCoTriangulate will halt.

When SafeCoTriangulate halts, Requirement 1 has been explicitly verified. □

By Theorems 11, 12 and 13, we conclude that the problem of cooperative triangulation with Requirements 1 through 3 is solved by SafeCoTriangulate.

9 Experimental Study

So far, we have shown that SafeCoTriangulate correctly solves the problem of cooperative triangulation. However, our analysis says nothing about how sparse the resultant triangulation is. In fact, DepthFirstEliminate specifys only the (partial) order of elimination in terms of subsets of nodes in a graph. The order of elimination within each subset is left unspecified. Depending on the (total) order used, different fill-ins can be produced.

In general, we prefer elimination orders that produce the minimum number of fill-ins. As it is NP hard [10] to find such orders, heuristics should be used. A simple and effective heuristics for a centralized triangulation is eliminating the node with the smallest fill-ins [2]. We have adopted the same heuristics to supplement the partial order specified in DepthFirstEliminate. To evaluate how well SafeCoTriangulate (supplemented by the heuristics) performs, we conducted the following experimental study:

We have implemented our algorithms in WEBWEAVR-III (the successor of WEBWEAVR-II written in JAVA). To evaluate the sparseness of results from cooperative triangulation, we compare with the results from the centralized triangulation using the same heuristics.

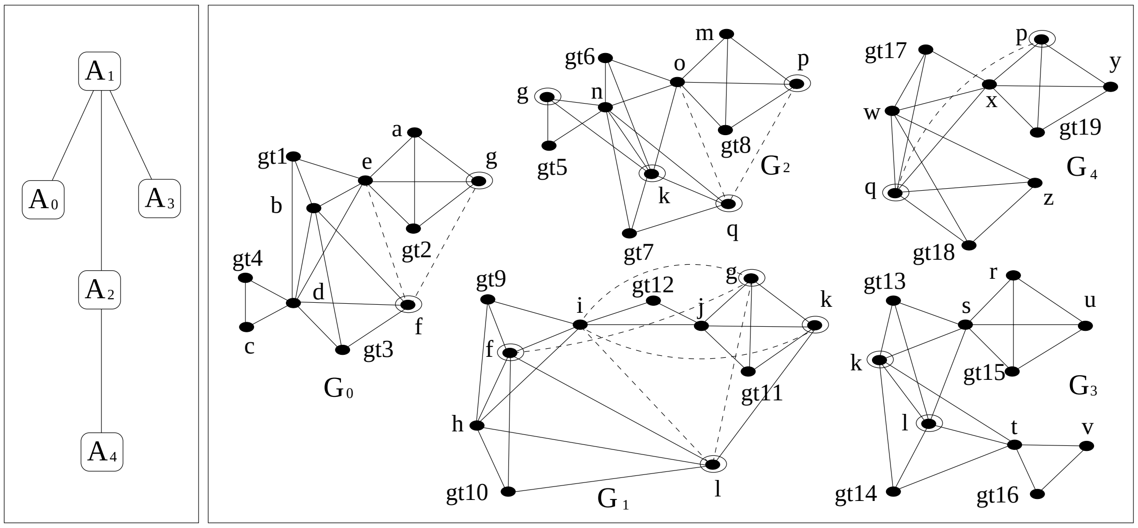

Figure 11: CIRCS: an experimental problem for cooperative triangulation.

Tests were run using three generated MSBNs: 5PARTS (not shown), CIRCS (Figure 11) and BIGB (Figure 12). The hypertree of CIRCS with five agents is shown in Figure 11 (left). Local graphs after moralization are shown in the right (without the dashed lines). Fill-ins added during cooperative triangulation are drawn as dashed lines. The result for BIGB is shown in Figure 12. For all three MSBNs, SafeCoTriangulate terminated with just one execution of CoTriangulate.

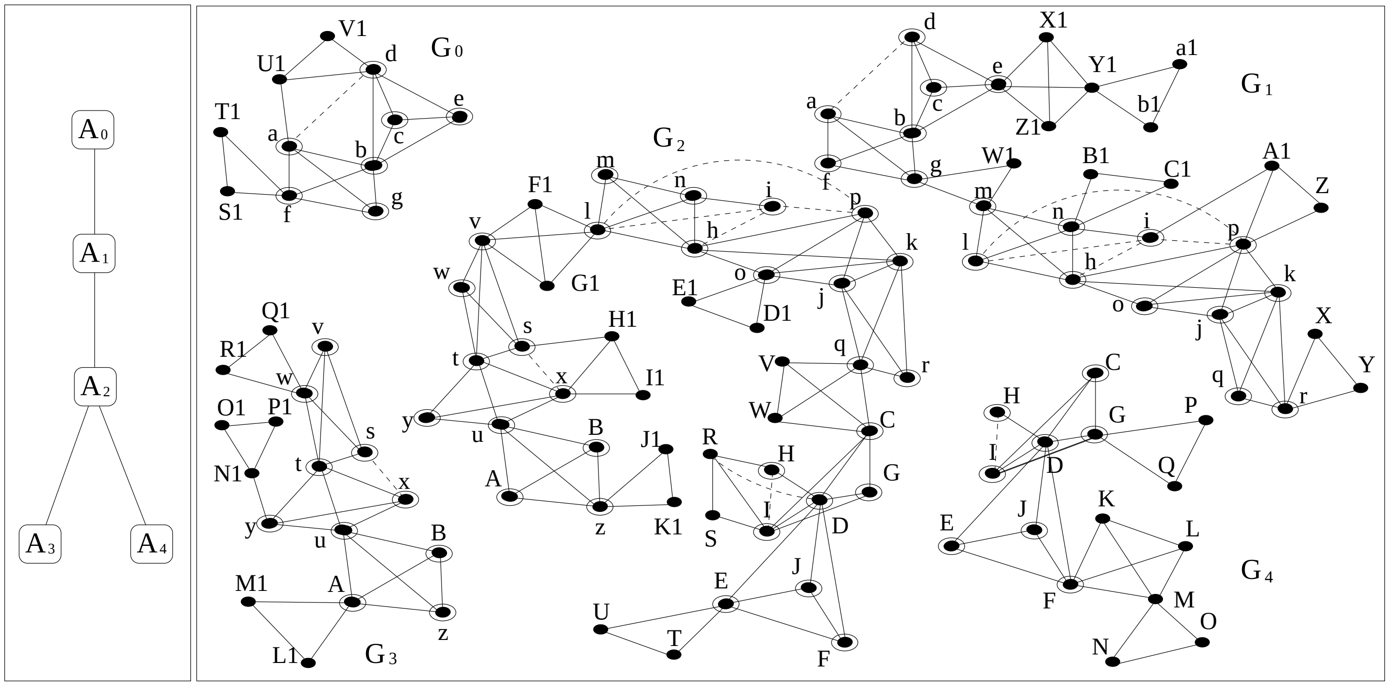

Figure 12: BIGB: an experimental problem for cooperative triangulation.

The experimental results are summarized in Table 1. The second column lists the total variables (shared variables are counted once) in each MSBN. The number of fill-ins produced by each centralized triangulation is listed in the third column, and the corresponding result for cooperative triangulation is listed in the last column.

For 5PARTS, a small MSBN, the same fill-in was produced by both methods. For CIRCS, each method produced eight fill-ins but two fill-ins were different. For BIGB, results from the two methods shared only two fill-ins, and the cooperative triangulation produced two more.

MSBN	Total number of variables	Fill-ins (centralized)	Fill-ins (cooperative)
5PARTS	21	1	1
CIRCS	45	8	8
BIGB	80	6	8

Table 1: Summary of experimental results

The results demonstrate that SafeCoTriangulate produces reasonably sparse triangulations compared with the centralized triangulation.

10 Complexity analysis

First, we analyze the time complexity of CoTriangulate. We concentrate on DepthFirstEliminate and focus on its elimination processing only, as the amount of computation in communication of fill-ins to adjacent agents during DepthFirstEliminate and DistributeDlink is minor.

Given a hypertree of n agents, from Figure 9, it's easy to see that $2n$ eliminations are performed during DepthFirstEliminate.

Let k be the maximum number of nodes in a local graph, and i be the maximum degree of a node. To eliminate a node, the completeness of its adjacency is checked. The complexity of the checking is $O(i^2)$. Using the heuristics, $O(k)$ nodes are checked before one is eliminated. Hence the time complexity of eliminating all nodes in a graph is $O(k^2\ i^2)$.

The complexity of CoTriangulate is then $O(n\ k^2\ i^2)$.

Next, we consider the complexity of SafeCoTriangulate. Clearly, the worst case complexity of SafeCoTriangulate is much higher than CoTriangulate. However, our experimental study provides evidence that the average case complexity of SafeCoTriangulate will be close to that of CoTriangulate.

Furthermore, efficiency of SafeCoTriangulate can be improved in several ways: For instance, if a d-sepset becomes complete during triangulation, then the hyperlink it represents is "blocked" and an entire sub(hyper)tree in one side of the hyperlink needs not be processed in later iterations of CoTriangulate if no fill-ins are produced in the subtree.

Another condition for early partial termination is the change of local topology. If an agent neither produces any fill-ins locally nor receives any from adjacent agents during one round of CoTriangulate, it does not have to participate in the next round (although it may have to participate in later rounds).

Based on these possible improvements and our empirical results, we believe that although the worst case complexity of SafeCoTriangulate is quite expensive, only a very small number of (often one) iterations of CoTriangulate need to be performed in many MSBNs. Most agents need not participate in most of the iterations of CoTriangulate except the first one. Therefore, the average case complexity of SafeCoTriangulate (with these improvements) will be similar with that of CoTriangulate.

11 Conclusion

Inference with a multi-agent MSBN can be performed effectively in a compiled representation. The compilation without compromising agents' privacy requires cooperative triangulation of a graph union without revealing each agent's local graph beyond the subgraph over shared nodes. We have proposed a conceptually simple and efficient algorithm (SafeCoTriangulate) for cooperative triangulation of a graph defined as the union of a set of graphs organized into a hypertree. The algorithm has been implemented and a small scale test demonstrated sparse triangulations compared with the centralized processing.

Acknowledgements

This work is supported by the Research Grant OGP0155425 from the Natural Sciences and Engineering Research Council (NSERC) of Canada. Helpful feedback were received on an early version from F. Jensen, U. Kjaerulff and K.G. Olesen.

Appendix: Other Graph-theoretical terminologies

A junction tree (JT) T over a set N is a tree where each node is labeled by a subset (called a *cluster*) of N and each link is labeled by the intersection (called a *sepset*) of its incident clusters, such that the intersection of any two clusters is contained in every sepset on the path between them. Two simple JTs are shown in Figure 4 (right).

A maximal complete set of nodes in a graph is called a *clique*. Given a chordal graph G over a set N of nodes, a JT T of G is created by labeling each node of T with a clique of G. Such a JT exists iff G is chordal. The two JTs in Figure 4 (right) are JTs of G and Q in Figure 4 (left), respectively.

For disjoint subsets X, Y and Z of nodes in a graph G, we use $< X|Z|Y >_G$ to denote that nodes in Z *graphically separate* nodes in X and nodes in Y. Graphical separation may be defined differently in different types of graphs. If G is an undirected graph, then the normal graphical separation applies. In Figure 4 (left), $\{a, d\}$ and $\{c, e\}$ are separated by $\{b\}$ in G. In a JT T over N, graphical separation is defined as follows: Let X, Y and Z be disjoint subsets of N. Suppose that for each $x \in X$ and each cluster C_x such that $x \in C_x$, and for each $y \in Y$ and each cluster C_y such that $y \in C_y$, there exists a sepset S on the path between C_x and C_y such that $S \subseteq Z$. Then X and Y are said to be graphically separated by Z in T. In Figure 4 (right), $\{a, d\}$ and $\{c, e\}$ are separated by $\{b\}$ in the left JT.

Let N be the set of variables in a domain and $P(N)$ be the probability distribution over N. For three disjoint subsets X, Y and Z of variables, X and Y are *conditionally independent* given Z if $P(X|Y, Z) = P(X|Z)$ *whenever* $P(Y, Z) > 0$. Denote the relation by $I(X, Z, Y)$. A graph G is an *I-map* [4] of a PDM over N if there is an one-to-one correspondence between nodes of G and variables in N such that for all disjoint subsets X, Y and Z of N, $< X|Z|Y >_G \implies I(X, Z, Y)$.

References

[1] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[2] F.V. Jensen. *An introduction to Bayesian networks.* UCL Press, 1996.

[3] D. Koller and A. Pfeffer. Oject-oridented Bayesian networks. In D. Geiger and P.P. Shenoy, editors, *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pages 302–313, Providence, Rhode Island, 1997.

[4] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[5] Y. Xiang. Optimization of inter-subnet belief updating in multiply sectioned Bayesian networks. In *Proc. 11th Conf. on Uncertainty in Artificial Intelligence*, pages 565–573, Montreal, 1995.

[6] Y. Xiang. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence*, 87(1-2):295–342, 1996.

[7] Y. Xiang. Belief updating in MSBNs without repeated local propagations. Technical Report CS-98-01, University of Regina, 1998.

[8] Y. Xiang, B. Pant, A. Eisen, M. P. Beddoes, and D. Poole. Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 5:293–314, 1993.

[9] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply sectioned Bayesian networks and junction forests for large knowledge based systems. *Computational Intelligence*, 9(2):171–220, 1993.

[10] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. of Algebraic and Discrete Methods*, 2(1), 1981.