Agent Interface Enhancement: Making Multiagent Graphical Models Accessible

Yang Xiang
University of Guelph, Canada
yxiang@cis.uoguelph.ca

Kun Zhang
University of Guelph, Canada
zhangk@uoguelph.ca

ABSTRACT

Multiagent probabilistic reasoning with multiply sectioned Bayesian networks requires interfacing agent subnets (the modeling task) subject to a set of conditions. To specify the interfaces such that they are both valid and efficient is non-trivial. We present multiagent interface enhancement that relieves agent designers from the burden of model efficiency so that multiagent graphical models become more accessible.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms

Keywords

Agent-Based Modeling, Cooperation and Coordination Among Agents, Graphical Models, MSBNs, Multiagent Architectures, Uncertain Reasoning

1. INTRODUCTION

Built upon the success of Bayesian Networks (BNs) [4], multiply sectioned Bayesian networks (MSBNs) [7] provide a rigorous framework for reasoning about uncertain domains in cooperative multiagent systems (MASs). Each agent holds its partial perspective (a subnet over a subdomain) of a domain, reasons autonomously as well as through limited communication with other agents. From a few high level requirements, (1) exact probabilistic measure of agent belief, (2) agent communication by belief over small sets of shared variables, (3) a simpler agent organization, (4) DAG domain structuring, and (5) joint belief admitting agents' belief on private variables and combining their beliefs on shared variables, it has been shown [9] that the resultant representation of a cooperative multiagent system is an MSBN.

To ensure autonomous, exact and efficient inference, subnets in an MSBN must satisfy a set of representational constraints. This is the modeling task. Specification of such a multiagent graphical model that is both valid and efficient is non-trivial and places a major burden on MAS designers. A key factor in modeling is the specification of agent interfaces, which are subject to hypertree, d-sepset, privacy, and sparse linkage tree conditions (to be introduced below). The main contribution of this work is a method to automate interface modeling so that MAS designers are relieved from the burden of model efficiency (the sparse linkage tree condition) and partially from the burden of privacy. The direct consequence is the simplification of modeling task and the improved accessibility of the MSBN framework to MAS designers.

2. THE ENHANCEMENT APPROACH

Before agents can aid graphical modeling, the MAS has to be created. We assume that for any large domain where an MAS is to be built, there is a natural subdomain division reasonably obvious to the human agent designers. The natural division divides the domain into overlapping subdomains, one for each agent, and thus forms the initial agent interfaces. For instance, in equipment diagnosis [7], each subsystem forms a natural subdomain and its input from and output to another subsystem form the interface between the two. In supply chain collaborative design [8], each component to be designed by a supplier forms a natural subdomain. The initial interfaces, however, may not support efficient inference. We let agents cooperate to enhance interfaces to improve efficience. Not requiring initial interfaces to be efficient removes the burden from agent designers and eases their modeling task.

One of the advantages of the MSBN framework is that once the modeling of MSBN-based MAS is complete, internal details of each agent (its subnet) is kept private throughout model verification, model compilation, and run-time inference. As we are now dealing with multiagent modeling, it is necessary to relax this privacy constraint *slightly* (but not completely). Each subnet is built by its agent designer. We assume that the designer can classify variables in the subnet into three sets: private, public, and preferably private. The *private* set should be kept so absolutely. The *public* set is included in the initial agent interface. The *preferably private* set is initially private, but the agent is allowed to make some elements public if it believes that the disclosure improves inference efficience. The agent is required, however, to keep the disclosed subset as small as possible.

We argue that the three-set assumption respects privacy realistically. Consider tightened airport security check after 9/11. Travelers who are subject to much more thorough body search can be regarded as giving up more privacy than before. However, many accept this as a price paid for a safer flight. The three-set assumption treats privacy as a form of utility and efficience as another, and allows them to be traded as desired by agent designer.

Furthermore, consider what would happen if human agent designers, instead of agents, are trying to construct efficient agent interfaces. For a complex domain, it is unlikely for each designer to classify its subdomain into private and public variables, and the resultant agent interfaces will be automatically both valid and efficient. More likely, they would start with initial interfaces and incrementally modify them, which effectively discloses some initially private variables to other agent designers. The three-set assumption enables this incremental process to be mirrored by agents.

We are unaware of other work on improving agent interfaces for multiagent graphical models. The most related are the information bottleneck (IB) approach [6] and its extension [1]. IB constructs a set T of variables that are stochastic functions of one set Y of variables and provide information on another set X. In general, T *approximates* information in X. Our enhanced interface conveys *exactly* all relevant information from one agent to another.

3. OVERVIEW OF MSBNS

An MSBN M is a collection of Bayesian subnets, one from each agent, representing probabilistic dependence of a *domain* divided into multiple *subdomains*. Agents cooperate to reason about the state of the domain [7]. To ensure distributed, exact inference, subnets satisfy conditions described below:

Let $G_i = (N_i, E_i)$ $(i = 0, 1)$ be two graphs. G_0 and G_1 are *graph-consistent* if subgraphs of G_0 and G_1 spanned by $N_0 \cap N_1$ are identical. Given graph-consistent graphs $G_i = (N_i, E_i)$ $(i = 0, 1)$, the graph $G = (N_0 \cup N_1, E_0 \cup E_1)$ is the *union* of G_0 and G_1, denoted by $G = G_0 \sqcup G_1$. Given a graph $G = (N, E)$, a division of N into N_0 and N_1 such that $N_0 \cup N_1 = N$ and $N_0 \cap N_1 \neq \emptyset$, and subgraphs G_i of G spanned by N_i $(i = 0, 1)$, G is *sectioned* into G_0 and G_1. Sectioning is used to relate graphical models at individual agents in a cooperative multiagent system.

DEFINITION 1. *Let $G = (N, E)$ be a connected graph sectioned into subgraphs $\{G_i = (N_i, E_i)\}$. Let the subgraphs be organized into an undirected tree Ψ where each node is uniquely labeled by a G_i and each link between G_k and G_m is labeled by the non-empty* `interface` *$N_k \cap N_m$ such that for each i and j, $N_i \cap N_j$ is contained in each subgraph on the path between G_i and G_j in Ψ. Then Ψ is a* `hypertree` *over G. Each G_i is a* `hypernode` *and each interface is a* `hyperlink`*. A pair of hypernodes connected by a hyperlink is said to be* `adjacent`.

Each hyperlink is an agent *interface*. Agents communicate by exchanging beliefs over their interfaces. An interface must be a *d-sepset*:

DEFINITION 2. *Let G be a directed graph such that a hypertree over G exists. A node x contained in more than one subgraph with its parents $\pi(x)$ in G is a* `d−sepnode` *if there exists at least one subgraph that contains $\pi(x)$. An interface I is a* `d−sepset` *if every $x \in I$ is a d-sepnode.*

The overall structure of an MSBN is a hypertree MSDAG:

DEFINITION 3. *A* `hypertree MSDAG` *$\mathcal{G} = \bigsqcup_i G_i$, where each G_i is a DAG, is a connected DAG such that (1) there exists a hypertree Ψ over $\mathcal{G}$, and (2) each hyperlink in Ψ is a d-sepset.*

Graphically, a hyperlink separates the hypertree MSDAG into two subtrees. Semantically, this corresponds to conditional independence given the d-sepset. An MSBN is defined as follows with an example shown in Figure 1.

DEFINITION 4. *An MSBN M is a triplet $M = (\mathcal{N}, \mathcal{G}, \mathcal{P})$. $\mathcal{N} = \bigcup_i N_i$ is the* `domain` *where each N_i is a set of variables. $\mathcal{G} = \bigsqcup_i G_i$ (a hypertree MSDAG) is the* `structure` *where nodes of each DAG G_i are labeled by elements of N_i. Let x be a variable and $\pi(x)$ be all the parents of x in $\mathcal{G}$. For each x, exactly one of its occurrences (in a G_i containing $\{x\} \cup \pi(x)$) is assigned $P(x|\pi(x))$, and each occurrence in other DAGs is assigned a constant table. $\mathcal{P} = \prod_i P_i(N_i)$ is the* `jpd`*, where each $P_i(N_i)$ is the product of probability tables associated with nodes in G_i. Each triplet $S_i = (N_i, G_i, P_i)$ is called a* `subnet` *of M. Two subnets S_i and S_j are said to be* `adjacent` *if G_i and G_j are adjacent on the hypertree MSDAG.*

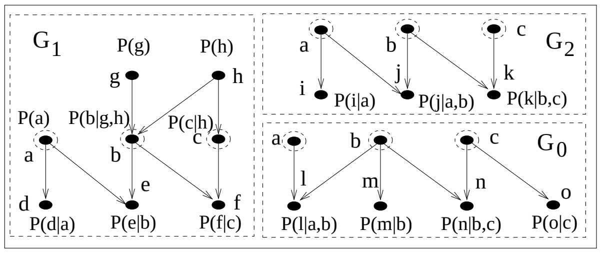

Figure 1: A trivial MSBN where each d-sepnode is shown with a dashed circle. The hypertree has the structure $G_1 - G_0 - G_2$ and each d-sepset is $\{a, b, c\}$.

To compute intra-agent messages effectively, each agent compiles its subnet into a junction tree (JT) [2], where variables are grouped into *clusters* with intersection of adjacent clusters referred to as *separators*. To integrate computation of intra-agent messages with computation of inter-agent messages seemlessly, the compilation is performed cooperatively. A key step is *cooperative triangulation* [7], during which each agent performs *constrained triangulation* through node elimination in partial order: A set N of nodes is eliminated according to a partial order (N_1, N_2) where $N = N_1 \cup N_2$, if nodes in N_1 are eliminated before nodes in N_2 are eliminated. We also refer to the triangulation as *constrained by* N_2, or *unconstrained* if $N_2 = \emptyset$.

For efficient computation of inter-agent messages, agents compile each d-sepset into a JT, called a *linkage tree*. See Figure 2 for linkage tree L_1 between T_0 and T_1. Each cluster in L_1 is called a *linkage*. Linkage $\{b, c\}$ is an message channel between cluster $\{b, c, f\}$ in T_1 and cluster $\{b, c, n\}$ in T_0. Its state space size (SSS) is 16 if the dimension (number of possible values) of variables a, b and c is 4, and that of L_1 is 32 which determines the efficience of inter-agent messaging. In general, the sparser the linkage tree (with many small linkages), the more efficient it is. If L_1 were made of a single linkage $\{a, b, c\}$, its SSS would be 64.

4. WHY NOT CENTRALIZE COMPILATION

An alternative to enhancing interfaces in MSBN is to build a centralized domain model, compile it into a JT, and separate the JT into subtrees one for each agent. Although it provides the most efficient run-time representation, it suffers from two major drawbacks compared to the approach we take.

First of all, the alternative requires that all domain variables and their dependence relations to be disclosed to the centralized model builder. There will be no agent privacy at all. In contrast, our approach does not disclose any private variables and dependence relations among them. It only discloses a subset of preferably private variables and does so in a cautious manner.

Secondly, even if the agent privacy were not a concern at all, the alternative will disrupt natural subdomain division among agents.

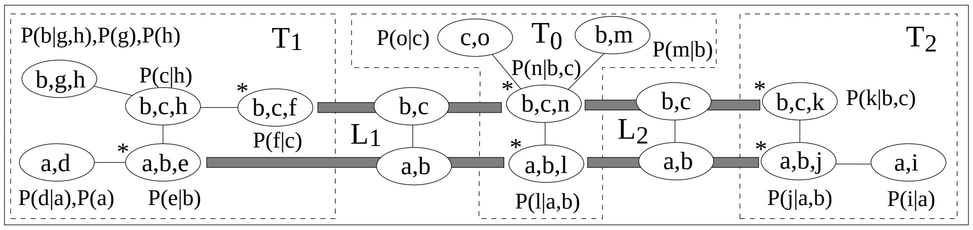

Figure 2: JTs and linkage trees obtained from Figure 1.

We illustrate this problem with a trivial MSBN shown in Figure 3. In (1), a trivial centralized domain model is shown as a BN. Suppose that it is naturally divided into two overlapping subdomains as shown in (2), where variables b, c and d form the interface. The BN in (1) is compiled into JT in (3). If this JT is to be separated into two, there are five possible ways, each separating the JT into two subtrees along one of the five separators (e.g., the separator between clusters $\{b, c\}$ and $\{b, f\}$ is $\{b\}$). None of them corresponds to the natural subdomains $\{a, b, c, d, e\}$ and $\{b, c, d, f, g\}$. The agents thus constructed lose their semantic coherence (the need for such coherence is also touched upon in [3]). On the other hand, according to the MSBN framework, the subdomain models in (2) will each be compiled into a JT in (4), which preserves the natural division while allowing efficient inference.

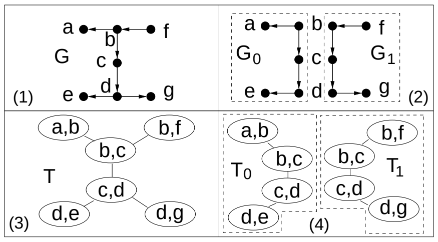

Figure 3: Illustration of natural subdomain disruption.

5. ANALYSIS

Inference in MSBNs is efficient mostly because agent interfaces, the d-sepsets, are represented as linkage trees, which enables potential factorization. Intuitively, this means that d-sepnodes are dispersed among a number of clusters in the local JT. Hence, the more scattered the d-sepnodes among clusters, the more efficient the inference. In the following, we analyze how adding more d-sepnodes can achieve such effect. Proposition 1 shows under what condition adding a node to d-sepset can disperse d-sepnodes into different clusters.

PROPOSITION 1. *Let S be the subnet of an agent, T be its local JT from S, and x and y be d-sepnodes contained in a cluster C in T. If a unique path exists between x and y in S, all other nodes on the path are non-d-sepnodes, and one of them, v, is head-to-tail or tail-to-tail, then adding v to d-sepset can disperse x and y into different clusters, unless it is prevented by other agents.*

Proof: Since the path contains at least v and it is head-to-tail or tail-to-tail, x and y will not be connected during moralization. Once v is added to the d-sepset, during constrained triangulation by the local agent, x can be eliminated before y and v, or y can be eliminated before x and v (neither is possible if v is not added to the d-sepset). In either case, x and y will not be connected by any fill-in. Hence, unless a moral link or a fill-in between x and y is added by another agent, x and y cannot be in the same cluster in the new local JT. $\square$

The only way the condition in Proposition 1 can be violated is where the unique path between x and y consists of a single head-to-head node v. If any arc on the path is reversed or any additional nodes exist on the path, the condition in Proposition 1 will be true. Because a moral link will connect x and y, no matter what the rest of the d-sepset is, x and y must be contained in the same cluster. This means that Proposition 1 captures *all* possible cases where a non-d-sepnode can be successfully added to d-sepset to disperse x and y.

The following corollary generalizes Proposition 1 to the case where multiple paths exist between x and y in the local subnet.

COROLLARY 1. *Let S be the subnet of an agent, T be its local JT from S, and x and y be d-sepnodes contained in a cluster C in T. For each path between x and y in S, if all other nodes on the path are non-d-sepnodes, and one of them, v, is head-to-tail or tail-to-tail, then adding such v from each path to d-sepset can disperse x and y into different clusters, unless it is prevented by other agents.*

When d-sepnodes are dispersed into different clusters, they are effectively dispersed into different linkages in the linkage tree. On one hand, this reduces sizes of corresponding linkages. As individual linkages become smaller, the state space of the linkage tree is reduced and hence the improved efficience. On the other hand, according to Corollary 1, dispersing d-sepnodes requires enlarging the d-sepset. The new d-sepnodes increase the state space in each linkage they participate. To achieve the net effect of reduction in linkage tree state space, new d-sepnodes must be chosen carefully.

In general, when addition of a single non-d-sepnode disperses multiple pairs of d-sepnodes, the negative effect of the addition is smaller than its positive effect and the linkage tree state space is decreased.

The above analysis has emphasized the topological aspect of the issue. However, since it is the total state space of the linkage tree that directly affects inference efficience, not cardinalities of linkages, another important factor is the dimensions of variables involved. When a linkage of three variables each of dimension 3 is dispersed, by adding a binary variable v to d-sepset, into three linkages each made of v and another variable (see Figure 5 (d) later for an example), SSS is reduced from 3*3*3 = 27 to 3*2*3=18. However, if v has a dimension of 4, SSS is increased to 3*4*3 = 36.

In general, when a new d-sepnode v disperses existing d-sepnodes whose dimensions are higher than that of v, the linkage tree SSS is decreased.

6. SEARCH FOR ENHANCEMENT

To improve interface efficience, we add non-d-sepnodes to a d-sepset to make the linkage tree sparser and reduce its SSS. We refer to a non-empty set of non-d-sepnodes in a subdomain as an *enhancement*, whose effectiveness depends also on enhancements from other agents. Hence, the optimal set of enhancements (two per d-sepset), which reduces the sum of all linkage tree state space sizes to the maximal extent, generally can only be obtained by exhaustive multiagent search. If each agent has $O(k)$ (where k is very large as shown below) alternative enhancements for each of its d-sepsets, then the search space of n agents has a complexity of $O(k^{2n})$. In the following, we consider several approximations with reduced complexity.

1. Each pair of adjacent agents performs cooperative search over the space of all pairs of enhancements (one from each agent), and then adds the pairwise top enhancement to their d-sepset.
2. Each agent searches for local top enhancements (there may be ties). Cooperative search is then performed by all agents over the space of all combinations of local top choices.
3. Each agent searches for local top enhancements. Then each pair of adjacent agents searches for pairwise top enhancements over the space of all pairs of local top choices.

The first approximation restricts cooperative search to pairwise. The computation has a complexity of $O(n\ k^2)$. The second approximation first reduces the number of alternative enhancements at each agent per each d-sepset from k to a few, say, k'. Then global cooperative search is performed with $O(k'^{2\ n})$ complexity. The third approximation has the lowest complexity of $O(n\ (k + k'^2))$. In this work, we study the effectiveness of this approximation and will investigate the other approximations in future research.

The top level algorithm below is run by the MAS coordinator to activate cooperative interface enhancement.

ALGORITHM 1 (COENHANCEINTERFACE).

```
choose an agent $A_*$ arbitrarily;
call $A_*$ to run EnhanceInterface with null parameters;
```

The following recursive algorithm is run by each agent. Without losing generality, we denote the agent executing the algorithm as A_0. The execution is activated by a caller, denoted as A_c, which is either an adjacent agent of A_0 or the MAS coordinator. If A_0 has additional adjacent agents, they are denoted $A_1, ..., A_m$. The subdomains of $A_c, A_0, ..., A_m$ are $N_c, N_0, ..., N_m$, respectively, their subnets are $G_c, G_0, ..., G_m$, the interface between A_0 and A_c (if an agent) is denoted D_c, and the interfaces between A_0 and $A_1, ..., A_m$ are denoted $D_1, ..., D_m$. The caller A_c passes two parameters into A_0: $NewD_c$ - a set of new d-sepnodes and $Arcs_c$ - a set of arcs (each either connecting elements in $NewD_c$ or connecting an element of $NewD_c$ with an existing d-sepnode). In return, A_0 will send parameters $NewD_c^{\rightarrow}$ and $Arcs_c^{\rightarrow}$ to A_c, where the arrow denotes output by A_0.

ALGORITHM 2 (ENHANCEINTERFACE). *When A_0 is called by A_c with parameters $NewD_c$ and $Arcs_c$, it does the following:*

```
compile $G_0$ into JT $T_0^*$ by unconstrained triangulation;
for $i = 1, ..., m$ and $i = c$ if $A_c$ is an agent, do
  compile $G_0$ into JT $T_i$ through triangulation constrained by
    $(N_0 \setminus D_i, D_i)$;
  perform $UpdateDsepset(G_0, D_i, T_i, T_0^*)$ and denote return
    value by $NewD_i^{\rightarrow}$;
  denote arcs in $G_0$ relevant to $NewD_i^{\rightarrow}$ as $Arcs_i^{\rightarrow}$;
for $i = 1, ..., m$, do
  call $A_i$ to EnhanceInterface with $NewD_i^{\rightarrow}$ and $Arcs_i^{\rightarrow}$;
  receive $NewD_i$ and $Arcs_i$ from $A_i$;
for $i = 1, ..., m$ and $i = c$ if caller is an agent, do
  enhance $D_i$ with $NewD_i$ and $NewD_i^{\rightarrow}$ if both are non-empty;
  update $G_0$ with $NewD_i$ and $Arcs_i$;
if $A_c$ is an agent, send $NewD_c^{\rightarrow}$ and $Arcs_c^{\rightarrow}$ to $A_c$;
```

EnhanceInterface consists of three *for* loops. In the first *for* loop, local subnet is compiled into a JT by constrained triangulation relative to each d-sepset. Using this JT and that obtained from unconstrained triangulation, the local top d-sepset enhancement is obtained by UpdateDsepset (see below for more on this step). Through the second *for* loop, the enhancement operation is propagated along the hypertree organization of the MAS, until it reaches agents located at the leaves of the hypertree. The results of local search by UpdateDsepset will then be propagated back towards A_*, through the last statement of the algorithm. In the third *for* loop, the agent enhances the d-sepsets and its local subnet.

The return value $NewD_i^{\rightarrow}$ from local search UpdateDsepset can contain an unique top enhancement. It can also contain a set of top enhancements (that tie). In that case, the search strategy 3 dictates a pairwise cooperative search, which can be performed by modifying the above algorithm as follows:

It may appear that the task to decide which pair of local top enhancements are pairwisely best requires cooperation of both agents. We show below that a simpler operation suffices.

We consider agent A_0 and its caller agent A_c. At the end of the first *for* loop of EnhanceInterface, A_0 has received $NewD_c$ and computed $NewD_c^{\rightarrow}$. For each enhancement in $NewD_c$ and each in $NewD_c^{\rightarrow}$, A_0 can perform constrained triangulation using the corresponding enhanced d-sepset. A linkage tree will then be derived. According to [7] [1], this linkage tree is equivalent to what would be derived by A_c. Hence, the best pair of enhancements can be computed by A_0 only, with the following statements added at the end of the first *for* loop:

```
search for best pair of enhancements with $A_c$ based on
  $NewD_c$ and $NewD_c^{\rightarrow}$, breaking ties arbitrarily;
update $NewD_c$ and $NewD_c^{\rightarrow}$ accordingly by deleting other
  enhancements;
```

To inform A_c of the result, the last statement of the algorithm needs to be modified into the following:

```
if $A_c$ is an agent, send $NewD_c^{\rightarrow}$ and $Arcs_c^{\rightarrow}$ to $A_c$ as well as the
  pairwisely best enhancement in $NewD_c$;
```

7. LOCAL HEURISTIC SEARCH

For an agent to search for the locally best enhancement to a d-sepset, the alternatives must be identified. This can be done by examining the local subnet or the local JT. Since linkage tree is generated from local JT and preserves conditional independence encoded in the JT (Proposition 7.6 [7]), we identify enhancements based on the local JT.

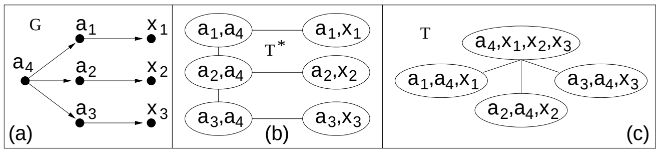

Figure 4: (a) Subnet G. (b) The JT T^* from G by unconstrained triangulation. (c) The JT T from triangulation constrained by $\{x_1, x_2, x_3\}$.

Consider the trivial subnet in Figure 4 (a). Given the d-sepset $\{x_1, x_2, x_3\}$, the local JT in (c) can be constructed. From the local JT, the linkage tree can be derived that has a single linkage $\{x_1, x_2, x_3\}$.

[1]Section 7.6.2 on two-agent cooperative triangulation and Section 8.3 on equivalence of linkage trees derived by adjacent agents.

To identify d-sepset enhancements, we consider JT T^* in (b) obtained by unconstrained triangulation. It preserves all conditional independence relations in (a) while T in (c) does not. Hence, T^* provides hints as to what conditional independence relations of (a) that has been removed in (c) can be utilized to improve the efficience of the d-sepset and the resultant linkage tree.

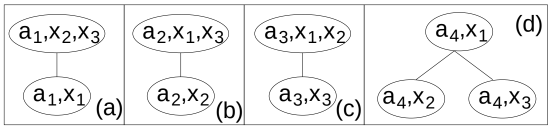

Figure 5: (a) The linkage tree obtained by adding a_1 to the d-sepset. (b) Obtained by adding a_2. (c) Obtained by adding a_3. (d) Obtained by adding a_4.

For example, $\{a_1\}$ is a separator in T^* between clusters $\{a_1, a_4\}$ and $\{a_1, x_1\}$, and it separates the cluster that contains the d-sepnode x_1 from clusters that contain the rest of the d-sepset. This suggests that $\{a_1\}$ is an alternative enhancement. After adding it to the d-sepset, the new linkage tree is shown in Figure 5 (a), where x_1 is indeed dispersed into a different linkage from the rest of d-sepset.

Following the same idea, $\{a_2\}$, $\{a_3\}$, $\{a_4\}$ are each an alternative enhancement. The corresponding linkage trees are shown in Figure 5 (b), (c) and (d), respectively. We define alternative enhancement formally below.

DEFINITION 5. *Let G be a subnet over N, D be its d-sepset with an adjacent subnet, and T^* be a JT of G obtained from unconstrained triangulation. An* `alternative enhancement` *E of D is the union of a set (possibly empty) of separators of T^* such that $E \cap D = \emptyset$.*

Given T^* with q clusters, in general, it has $O(2^q)$ enhancements, where q is bounded by the number of subdomain variables. We explore two techniques to improve computational efficience: leaf cluster removal and topological dominance. Below, we present the local search algorithm and then analyze two techniques. All d-sepnodes and non-d-sepnodes are relative to the d-sepset D.

ALGORITHM 3 (UPDATEDSEPSET).

```
Input: a subnet $G$ over $N$ and its d-sepset $D$ with an adjacent
  subnet; a JT $T$ from $G$ by constrained triangulation with partial
  order $(N \setminus D, D)$; a JT $T^*$ obtained from $G$ by unconstrained
  triangulation;

perform the following recursively until no cluster can be deleted,
  for each leaf cluster $Q$ of $T^*$,
    if $Q$ contains no d-sepnodes, delete $Q$ from $T^*$;
    else if $Q$ is a leaf cluster in $T$ with the same separator as in $T^*$,
      delete $Q$ from $T^*$;
if T* contains no separator with preferably private nodes only,
  return $\emptyset$;

CanNewD = empty;
for each separator W of T* with preferably private nodes only, do
  add $\{W\}$ to CanNewD as an 1-separator member;
  copy T* to T';
  delete each occurrence of W in T' to split it into subtrees;
  if each subtree has a single cluster with d-sepnodes,
    label W as final;

k=1, more=true;
while more == true, do
  more = false;
  for each k-separator member M of CanNewD, do
    copy T* to T';
    for each separator R in M, do
      delete each occurrence of R in T' to split it into subtrees;
    if each subtree has a single cluster with d-sepnodes,
      label M as final;
    else
      for each 1-separator member $\{W\}$ of CanNewD, non-final,
          $W \notin M$, and $M \cup \{W\} \notin CanNewD$, do
        if there exists a subtree where W is a separator on the path
            between 2 clusters with d-sepnodes, do
          add $M \cup \{W\}$ to CanNewD as a k+1-separator member;
          more = true;
  k++;

for each member M of CanNewD, do
  perform constrained triangulation with M added to d-sepset;
  compute SSS $\alpha$ of the resultant linkage tree;
find member $M^*$ of CanNewD with the smallest $\alpha$ value $\alpha^*$;
$\beta$ = SSS of the linkage tree derived from $T$;
if $\beta - \alpha^*$ is large than a threshold, return $M^*$;
else return $\emptyset$;
```

UpdateDsepset consists of four segments (separated by blink lines). The first removes leaf clusters in T^*. Proposition 2 below shows that these clusters are irrelevant and hence their removal reduces the space of alternative enhancements. The analysis is based on h-separation [7] in JTs:

DEFINITION 6. *Let H be a JT over N, and X, Y, Z be disjoint subsets of N such that no $x \in X$ and $y \in Y$ are contained in the same cluster in H. For $x \in X$ contained in cluster Q_x and $y \in Y$ contained in cluster Q_y, x and y are h-separated by Z if there exists a separator $S \subseteq Z$ on the path between Q_x and Q_y. X and Y are h-separated by Z if for every $x \in X$ and $y \in Y$, x and y are h-separated by Z.*

For instance, in Figure 4 (2), $\{x_1\}$ is h-separated from $\{x_2, x_3\}$ by $\{a_1\}$, written as $< x_1|a_1|x_2, x_3 >_{T^*}$ or simply $< x_1|a_1|x_2, x_3 >$. We are interested in h-separation relations $< X|Z|Y >$ relative to a given d-sepset D, where $X \subset D$, $Y \subset D$ and $Z \cap D = \emptyset$. We refer to such a relation as a *h-separation relative to D*.

PROPOSITION 2. *Let N be the domain of T^* and E be an alternative enhancement containing separators in T^*. After recursive deletion (relative to d-sepset D) of leaf clusters as specified in UpdateDsepset, let the resultant JT be T' whose domain is $N' \subset N$ and $E' = E \cap N' \subset E$. Let H denote the set of h-separations relative to D implied by E and H' be that implied by E'. Then $H = H'$.*

Proof: Intuitively, E and E' are the same except that E contains additional separators in T^* that are deleted by UpdateDsepset. We show that leaf cluster deletion by UpdateDsepset does not remove h-separations implied by E.

Let Q be a leaf cluster in T^* with separator V. We denote $Q \setminus V$ by C. The only h-separations relative to D that Q represents are in the form of $< C^-|V|U^- >$, where C^- is any subset of C and U^- is any subset of $N \setminus Q$.

The h-separations relative to D implied by E are in the form $< X|Z|Y >$ where $X \subset D$, $Y \subset D$, and $Z \cap D = \emptyset$. If Q contains no d-sepnodes, then C^- contains no d-sepnodes. Hence, removal of Q cannot remove any h-separations implied by E.

Next, consider the case where Q contains d-sepnodes. If Q is a leaf cluster in T (as defined in UpdateDsepset) and its separator in T is also V, then the h-separations represented by Q are also true in T. This means that these relations are not implied by E. Hence, removal of Q cannot remove any h-separations implied by E.

Since no h-relations in H is removed by leaf cluster deletion in UpdateDsepset, we obtain $H = H'$. □

Proposition 2 implies that leaf cluster deletion by UpdateDsepset does not introduce approximation to local search while improving its efficience. Proposition 3 below establishes that the early termination of UpdateDsepset is a correct action.

PROPOSITION 3. *If UpdateDsepset returns $\emptyset$ at the end of first segment, then the d-sepset D cannot be improved.*

Proof: When $T*$ contains no separator with preferably private nodes only, no legal separator Z can be found such that $< X|Z|Y >$ holds, where $X \subset D$, $Y \subset D$, and $Z \cap D = \emptyset$. □

8. TOPOLOGICAL DOMINANCE

The second and third segments of UpdateDsepset apply the technique of topological dominance or t-dominance to further trim the local search space. The concept of dominance has been used in game-theoretic research, e.g., [5]. We apply here from a topological perspective. In Figure 5 (d), each original d-sepnode has been isolated in a separate linkage. This suggests that adding more nodes to d-sepset may enlarge the linkage tree SSS. In fact, adding to $\{a_4\}$ any variable of a dimension no less than that of a_4 will do poor than $\{a_4\}$. We say that any such superset is *t-dominated* by $\{a_4\}$.

On the other hand, the enhancement $\{a_1\}$ does not t-dominate every proper superset. For instance, in the linkage tree produced from $\{a_1\}$ (Figure 5 (a)), x_2 and x_3 are contained in the same linkage. In the linkage tree produced from the superset $\{a_1, a_2\}$ (Figure 6 (a)), x_2 and x_3 are dispersed into different linkages. Hence, enhancement $\{a_1, a_2\}$ is not t-dominated by $\{a_1\}$.

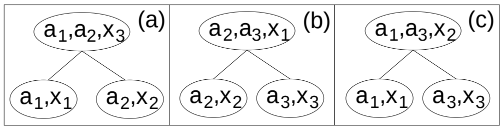

Figure 6: (a) The linkage tree obtained by adding $\{a_1, a_2\}$ to the d-sepset. (b) Obtained by adding $\{a_2, a_3\}$. (c) Obtained by adding $\{a_1, a_3\}$.

If an enhancement is not t-dominated by any proper subset, then it is called a *non-t-dominated* enhancement. For instance, $\{a_1, a_2\}$ is an non-t-dominated enhancement. If an non-t-dominated enhancement t-dominates every proper superset, it is called a *final* enhancement. Enhancement $\{a_4\}$ is final, but $\{a_1\}$ is not final.

Figure 6 shows linkage trees from three alternative enhancements each of which is obtained by including two separators in Figure 4 (b). All of them are non-t-dominated and final.

DEFINITION 7. *Let E be an alternative enhancement of a d-sepset D and H be the set of h-separations relative to D implied by E. Let $E' \supset E$ be another enhancement and H' the corresponding h-separations. E' is* $\mathtt{t-dominated}$ *by E, if for each h-separation $< X|Z'|Y >$ in H', there exists $< X|Z|Y >$ in H, such that $Z' \supseteq Z$. If there exists no $E' \subset E$ such that E' t-dominates E, then E is* $\mathtt{non-t-dominated}$. *If for all $E' \supset E$, E' is t-dominated by E, then E is* $\mathtt{final}$.

Given the four separators in Figure 4 (b), there are a total of 15 alternative enhancements. Linkage trees from seven are shown in Figs. 5 and 6. The remaining eight are t-dominated.

As assumed, each agent classifies subdomain variables into private, public, and preferably private. For instance, the agent could classify a_4 as private, but a_1, a_2, a_3 as preferably private. This classification trims the set of non-t-dominated enhancements. In particular, only enhancements corresponding to Figure 5 (a), (b), (c), and Figure 6 (a), (b), (c) are now valid. Proposition 4 below shows under what condition, ignoring t-dominated enhancements causes no loss of accuracy to local search. We omit proofs for the remaining formal results due to space.

PROPOSITION 4. *Let AE be the set of all alternative enhancements of a d-sepset D for a given subnet. Let ND be the set of non-t-dominated elements of AE. Let TOP_{AE} be the set of elements of AE whose linkage tree state space sizes are minimal among elements of AE, and TOP_{ND} be the corresponding set relative to ND. If dimensions of non-d-sepnodes are identical, then $TOP_{AE} = TOP_{ND}$.*

The second and third segments of UpdateDsepset compute non-t-dominant enhancements. According to Proposition 4, local search over these enhancements is only accurate when preferably private variables have identical dimensions. When this condition does not hold, local search restricted by t-dominance becomes a heristics and introduces approximation.

The second segment identifies 1-separator non-t-dominated enhancements. The third segment identifies non-t-dominated enhancements made of multiple separators. Note that the concept of final enhancement is utilized to skip t-dominated enhancements and improve efficience. The following proposition shows that *final* enhancements are correctly identified. It can be proven by a direct application of Def. 7.

PROPOSITION 5. *Let E be an enhancement and Ψ be the set of subtrees resultant from deletion of each separator of E in T'. If each subtree in Ψ has a single cluster with d-sepnodes, then E t-dominates every enhancement $E' \supset E$.*

The following theorem establishes the significance of final enhancements and functionality of the second and third segments.

THEOREM 1. *At the completion of the third segment of UpdateDsepset, $CanNewD$ contains all and only non-t-dominated enhancements relative to d-sepset D.*

After local search space has been reduced to non-dominated enhancements, can it be trimmed without introducing further approximation? The following example answers the question negatively.

For variables in Figure 4 (a), denote dimensions of $a_1, \cdots, a_4$ as $m_1, \cdots, m_4$, respectively, and those for $x_1, \cdots, x_3$ as $n_1, \cdots, n_3$, respectively. Denote by k_1 the SSS of linkage tree from constrainted triangulation with d-sepset $\{x_1, x_2, x_3\}$ (Figure 4 (c)). Denote the SSS of linkage trees shown in Figure 5 as k_2, k_3, k_4, k_5, respectively, and those in Figure 6 as k_6, k_7, k_8, respectively. Then $k_1, \cdots, k_8$ can be computed as follows:

$$\begin{aligned}
&k1 = n_1 * n_2 * n_3,\\
&k2 = m_1 * (n_2 * n_3 + n_1), \quad k3 = m_2 * (n_1 * n_3 + n_2),\\
&k4 = m_3 * (n_1 * n_2 + n_3), \quad k5 = m_4 * (n_1 + n_2 + n_3),\\
&k6 = m_1 * m_2 * n_3 + m_1 * n_1 + m_2 * n_2,\\
&k7 = m_2 * m_3 * n_1 + m_2 * n_2 + m_3 * n_3,\\
&k8 = m_1 * m_3 * n_2 + m_1 * n_1 + m_3 * n_3
\end{aligned}$$

Table 1: State space sizes $k_1, \cdots, k_8$ given variable dimensions. Upper half: variable dimensions. Lower half: state space sizes.

n_1	2	3	3	3	5	5	7	7
n_2	2	3	3	3	5	6	6	4
n_3	2	3	3	3	5	5	5	5
m_1	2	2	3	3	2	2	4	3
m_2	2	3	2	3	3	3	2	4
m_3	2	3	3	2	3	3	3	2
m_4	2	3	3	3	3	4	5	5
k_1	8	27	27	27	75	150	210	140
k_2	12	24	36	36	60	70	148	81
k_3	12	36	24	36	90	93	82	156
k_4	12	36	36	24	90	105	141	66
k_5	12	27	27	27	45	64	90	80
k_6	16	33	33	45	55	58	80	97
k_7	16	45	33	33	75	78	69	82
k_8	16	33	45	33	55	61	115	55

Table 1 shows values for $k_1, \cdots, k_8$ given dimensions $n_1, \cdots, n_3$, $m_1, \cdots, m_4$. The minimal linkage tree SSS for each row occurs at a different column. For instance, for the first row (with all variables binary), the optimal size is 8 and is located at the first column in the right half. For the second row, the optimal size 24 is located at the second column, and so on.

This example demonstrates that any non-t-dominated enhancement may be the best choice, given right variable dimensions. In order not to cause further approximation, all non-t-dominated enhancements must be evaluated. This is done in the fourth segment. For simplicity, we phrased return value as a single enhancement, although it can be multiple enhancements that tie or top the list.

9. SUBNET CONNECTEDNESS

When an agent A passes a set of new d-sepnodes to an adjacent agent A', it is possible that some new d-sepnodes are disconnected in the updated subnet of A'. These new d-sepnodes are dependent of the rest of the subdomain of A' (since they are connected to other d-sepnodes in the subdomain of A). Hence, their disconnectedness in the updated subnet of A' creates misconception of independence. They violate convention of graphical models where connection implies dependence and disconnection implies independence.

To avoid the misconception, for each new d-sepnode v, we need to ensure that, in the updated subnet of A', it is connected to at least one existing d-sepnode. When v is not already connected to existing d-sepnodes, arcs must be added to achieve the connection. The arcs added will make the new subnet denser, but should not make it unnecessarily so, because any increase in the density of the subnet will translate into an increase in the size of state space of the resultant linkage tree. Based on these criteria, we identify the following subtasks:

First, agent A needs to recognize when some new d-sepnodes are disconnected to existing d-sepnodes. A new d-sepnode v is connected to existing d-sepnodes if (1) v is directly connected to an existing d-sepnode, or (2) v is indirectly connected an existing d-sepnode through a path consisting entirely of new d-sepnodes. This subtask can be accomplished with the following algorithm:

ALGORITHM 4 (MARKCONNECTED).

```
*mark each existing d-sepnode;*
*perform the following recursively,*
  *for each marked node, do*
    *if it has any adjacent new d-sepnode $v$, mark $v$;*
```

The following proposition shows that any new d-sepnode left unmarked by MarkConnected is disconnected from the existing d-sepset. Its proof is trivial.

PROPOSITION 6. *After MarkConnected is performed, if there exist unmarked new d-sepnodes, then each such node is disconnected from the existing d-sepnodes.*

Second, once a disconnected new d-sepnode v is identified, agent A needs to propose arc(s) to be added, that is, which other d-sepnodes should v connect to and direction of arcs. We show below that inadequate arcs added may create either directed cycle or significantly increase the size of state space of the local JT.

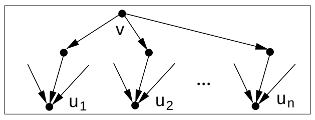

Figure 7: A subnet whose d-sepset with an adjacent subnet is $\{u_1, u_2, ..., u_n\}$.

In Figure 7, the existing d-sepset is $\{u_1, u_2, ..., u_n\}$. Enhancing the d-sepset with node v will significantly reduce the size of state space of the linkage tree. However, if v is added to the subdomain of the adjacent agent, it is disconnected. To make it connected to the existing d-sepset, it can be connected to a node u_i. If the arc (u_i, v) is added, it forms a directed cycle. On the other hand, if the arc (v, u_i) is added, the node u_i has one more parent. In the local JT, there is at least one cluster that contains u_i and all its parents. At least the state space of this cluster is enlarged by a factor of the dimension of v due to adding (v, u_i).

We propose a method for arc addition using the updated linkage tree in A. The following algorithm pairs a disconnected new d-sepnode with a connected d-sepnode (either existing or new).

ALGORITHM 5 (PAIRDISCONNECTED).

```
*Input: existing d-sepset $D$, enhanced d-sepset $D^+$, subset $X$ of $D^+$*
    *marked by MarkConnected, and $Y = D^+ \setminus X$;*

*get linkage tree $L$ with triangulation constrained by $D^+$;*
*find a cluster $C$ such that $C \cap Y \neq \emptyset$ and $C \cap X \neq \emptyset$;*
*pick $v \in C$ such that $v \in Y$ and $u \in C$ such that $u \in X$;*
*return pair $(v, u)$;*
```

We show that as long as $Y \neq \emptyset$, there always exists a cluster C as described in PairDisconnected, and hence the pair (v, u) can always be found.

PROPOSITION 7. *Let D be the existing d-sepset of a subnet with an adjacent subnet, D^+ be the enhanced d-sepset, X be the subset of D^+ marked by MarkConnected, and $Y = D^+ \setminus X \neq \emptyset$. Then, PairDisconnected will succeed and return (v, u).*

For each pair of d-sepnodes (v, u) identified, agent A notifies agent A'. In response, A' adds new d-sepnode v to its subdomain and adds a new private binary variable c as the common child of v and u. This method has several desirable properties:

First, clearly the new node v in A' is now connected in the updated subdomain of A'. Second, since c is a private node of A', there is no arc to be added in A relative to v. Hence, the state space of its local JT will not be compromised. Third, since c has only two adjacent nodes in A', it will cause the formation of a cluster

$\{c, v, u\}$ in the local JT of A'. Since the cluster has only three variables, it increases the state space of the local JT only slightly.

Disconnected new d-sepnodes may be divided into several groups where nodes in each group is connected to each other. In such case, A needs to run MarkConnected and PairDisconnected once for each group.

10. PROPERTIES OF ENHANCEMENT

We analyze key properties of CoEnhanceInterface. First, after CoEnhanceInterface, each agent interface is a d-sepset.

PROPOSITION 8. *Let E be an enhancement of d-sepset D from agent A relative to an adjacent agent A'. Then $D \cup E$ is a d-sepset.*

Second, CoEnhanceInterface preserves the hypertree condition.

PROPOSITION 9. *Let a d-sepset D between two adjacent subnets of an MSBN M be enhanced to D^+. Then hypertree condition holds in updated structure of M.*

Propositions 8 and 9 guarantee that the outcome of enhancement is a valid MSBN. Finally, all dependence relations in the original MSBN are preserved after agent interface enhancement.

PROPOSITION 10. *Let a d-sepset D between two adjacent subnets of an MSBN M be enhanced to D^+. If the structure of M is an I-map over N, then the updated structure of M is still an I-map over N.*

From Proposition 10, it follows that the enhanced MSBN preserves exact inference.

11. EXPERIMENT

To evaluate the effectiveness of the algorithms, we simulated 10 MSBNs, each with 5 subnets. Each subnet has between 50 to 75 nodes and each variable has a dimension up to 4. Each d-sepset has 8 variables.

For improvement of local search efficience, results of UpdateDsepset in 14 subnets are shown below. The total number of enhancements for each subnet ranges from $2^{24} > 1.6 \times 10^7$ to $2^{51} > 2 \times 10^{15}$. Due to leaf cluster removal and topological dominance, the number of non-t-dominated enhancements per subnet is between 31 and 2495.

Subnet	n_0	n_1	n_2	n_3	n_4	n_5	n_6
$\|CanNewD\|$	71	355	239	1063	791	35	363
# enhancement	2^{24}	2^{51}	2^{24}	2^{41}	2^{24}	2^{24}	2^{44}
Subnet	n_7	n_8	n_9	n_10	n_11	n_12	n_13
$\|CanNewD\|$	725	31	35	77	31	119	2495
# enhancement	2^{24}	2^{33}	2^{24}	2^{24}	2^{43}	2^{26}	2^{35}

For overall efficience improvement, results of CoEnhanceInterface with pairwise search are shown below. For each MSBN, SSS for each linkage tree is counted and sum over the four linkage trees is shown. All randomly selected d-sepsets have the worst complexity. Enhancement reduces the sum to between 44% and 94%. As the price, the largest number of preferably private variables disclosed to other agents occurred in an agent with 3 adjacent agents and a subdomain of 75 variables. It disclosed a total of 12 variables (an average of 4 per d-sepset).

MSBN	m_0	m_1	m_2	m_3	m_4
Before enhancement	82944	82944	82944	82944	82944
After enhancement	34286	12398	4562	7220	7020
MSBN	m_5	m_6	m_7	m_8	m_9
Before enhancement	82944	82944	82944	82944	82944
After enhancement	7720	46098	14540	25860	6894

In addition, we simulated a digital system of 216 devices. To monitor it, we built a MSBN model of 5 agents whose subdomains are sized at 91, 202, 72, 149 and 91 variables, respectively. The initial four agent interfaces have a total linkage tree SSS sum of 11264 and it is reduced to 728 (6.5%) through enhancement.

12. CONCLUSIONS

Graphical models such as MSBNs provide a framework for exact and autonomous probabilistic reasoning for multiagent systems. However, construction of such models that are both valid and efficient is involved. This contribution allows MAS designers to start with a model that is not necessarily efficient and let agents to improve its efficiency automatically. This approach relieves designers from the burden of model efficiency and makes the MSBN-based MAS more accessible. Through formal analysis and experiments, we have shown that the proposed agent interface enhancement explores only a small portion of the local search space and joint search space, while improving the model efficiency significantly.

13. ACKNOWLEDGEMENT

Financial support from NSERC of Canada and NSF of USA is acknowledged.

14. REFERENCES

[1] N. Friedman, O. Mosenzon, N. Slonim, and N. Tishby. Multivariate information bottleneck. In J.S. Breese and D. Koller, editors, *Proc. 17th Conf. on Uncertainty in Artificial Intelligence*, pages 152–161, San Francisco, 2001. Morgan Kaufmann.

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

[3] S.M. Mahoney and K.B. Laskey. Network engineering for complex belief networks. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 389–396, 1996.

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

[5] J.S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994.

[6] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In B. Hajek and R.S. Sreenivas, editors, *Proc. 37th Allerton Conf. on Communication, Control and Computation*, pages 368–377, 1999.

[7] Y. Xiang. *Probabilistic Reasoning in Multi-Agent Systems: A Graphical Models Approach*. Cambridge University Press, 2002.

[8] Y. Xiang, J. Chen, and W.S. Havens. Optimal design in collaborative design network. In *Proc. 4th Inter. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 241–248, 2005.

[9] Y. Xiang and V. Lesser. On the role of multiply sectioned Bayesian networks to cooperative multiagent systems. *IEEE Trans. Systems, Man, and Cybernetics-Part A*, 33(4):489–501, 2003.