Privacy Preserving Existence Recognition and Construction of Hypertree Agent Organization [*]

Yang Xiang and Kamala Srinivasan
School of Computer Science, University of Guelph, Canada

January 27, 2015

Abstract

Decentralized probabilistic reasoning, constraint reasoning, and decision theoretic reasoning are some essential tasks of cooperative multiagent systems. Several frameworks for these tasks organize agents into a junction tree (JT). We show that existing techniques for JT existence recognition and construction leak information on private variables, shared variables, agent identities and adjacency, that can potentially be protected. We present a scheme to quantify these privacy losses. We develop two novel algorithms for JT existence recognition and for JT construction when existing, that provide strong guarantee of agent privacy. Our experimental comparison shows that the proposed algorithms out-perform existing techniques, one of them having the lowest privacy loss and the other having no privacy loss, while being more efficient than most alternatives.

1 Introduction

Decentralized probabilistic reasoning, constraint reasoning, and decision theoretic reasoning (referred to as decision making below) are some essential tasks of cooperative multiagent systems (MAS). For simplicity, we refer to these tasks as *inference*. A number of alternative frameworks have been proposed for multiagent probabilistic reasoning [VKV02, Xia02], multiagent constraint reasoning [MB04, MSTY05, PF05, SF05, ZJSF08, VRAC10, BM10, XMZ14], and multiagent decision making [GD01, KM01, XH11]. Some frameworks do not assume a specific agent organizational structure [VKV02, KM01]. Some assume a total order among agents [MB04]. Some use a pseudotree organization [MSTY05, PF05, MSY08]. Some depend on a junction tree (JT) organization [VRAC10, BM10, Xia02, XMZ14, XH11]. A cluster in the JT may not have an internal structure [VRAC10, BM10] or each cluster may itself be organized into a local JT [Xia02, XMZ14, XH11], in which case the JT organization is called a *hypertree*. In this paper, we refer to JT organization and hypertree interchangeably. Hypertree is used to emphasize that clusters of the JT may be internally structured. JT is used when the topological nature (running intersection, defined below) of the organization is emphasized.

An effective agent organization not only needs to support inference but also agent privacy [SR04, SAZB05, SFP06, MPB$^+$06, GPBT06, FLP08, DMS$^+$08]. JT is one such organization. We show that a hypertree not only enables sound and efficient inference, but also has the potential for certain types of agent privacy. However, the potential does not ensure its delivery. We show that existing hypertree-based frameworks leak information on private variables, on shared variables, and on agent identities and adjacency, that can potentially be protected. Such compromise of privacy may limit adoption of these frameworks.

Consider trouble-shooting a complex piece of equipment (e.g., a chemical fertilizer plant) by an MAS, a task of multiagent probabilistic reasoning. The equipment consists of multiple components supplied by independent venders, who also supply computational agents each monitoring and trouble-shooting a component by cooperating with other agents. The agents can form a multiply sectioned Bayesian network (MSBN) [Xia02], where the core knowledge of each agent is encoded as a Bayesian

[*]This article significantly extends Xiang and Srinivasan [XS13b, XS13a] as part of the Canadian AI 2013 Proceedings.

subnet [Pea88] over many variables. Variables (nodes) in a subnet represent states of various devices and sensors in the component. When two components interact (e.g. the output of one feeds into the other), the subnets share variables that represent states of their interface. Knowledge on unique (private) variables in each subnet is typically proprietary to the component vendor. Knowledge on shared variables between subnets are often proprietary to the relevant component vendors. Knowledge on the identity of an agent and its interfacing relations, which translates into knowledge on existence of certain component and its interfacing relations, is often proprietary to the owner of the plant.

Suppose an agent honestly follows MSBN algorithms (therefore conducting trouble-shooting tasks as specified) and log messages exchanged during operation. These messages are sent to its vendor for purposes such as monitoring, bookkeeping, and maintenance. If these messages contain identities of other agents in the MAS, their interfacing relations, and variables in their subnets, then a significant amount of proprietary knowledge is leaked beyond their owners. Without strong guarantees against such loss of privacy, the plant owner would hesitate to adopt an MSBN-based MAS, and component vendors would hesitate to participate in such a MAS, in fear of losing their intellectual property. Similar situations exist in collaborative industrial design on supply chain [XCD04] and other applications.

A number of operations, including inference, are performed over the lifetime of a hypertree-based MAS. Among them, the most critical to agent privacy is hypertree construction. We identify three types of agent privacy that are naturally preserved during inference in such a MAS but are compromised by existing hypertree-based MAS frameworks. We propose a new algorithm for hypertree construction, if it exists, that significantly improves preservation of these types of privacy. We propose a second algorithm that recognizes hypertree existence while preserving privacy. We show that it can extend to construction as well. To the best of our knowledge, no known hypertree-based MAS frameworks provide the same degree of agent privacy during hypertree construction as the proposed algorithms.

Hypertree construction concerns three technical issues. The first determines whether a hypertree exists given the environment decomposition of an application. The second concerns how to construct a hypertree when one exists. The third involves how to modify an environment decomposition that has no hypertrees. This paper addresses the first two issues. For the third, we show that an environment decomposition without hypertrees can be modified to have one at the cost of a limited loss of privacy. How to make the modification while minimizing such loss is deferred to the sequel of this research.

Two fundamentally different approaches for privacy can be identified. One is to transmit private information into the public, but make it unintelligible to unintended receivers by encryption [YSH05, LOF10]. The other does not transmit private information in the first place. This is the approach taken in this work. We show that by carefully designing messages and message passing protocols, hypertree recognition and construction can be performed without leaking the three types of private information.

We assume that all agents honestly follow intended algorithms as in the above trouble-shooting example. This is a standard assumption in multiagent probabilistic reasoning and constraint reasoning, and in certain cooperative decision making. This assumption is justified by the fact that deviation from intended algorithms most likely lead to incorrect results, e.g., incorrect posterior probabilities in probabilistic reasoning and infeasible solutions in constraint reasoning. An agent that fails to perform its intended task will eventually be detected and abandoned. Hence, agents following the intended algorithms should be the norm. Under the assumption, this work does not address adversarial behaviors of agents. We present hypertree recognition and construction algorithms with strong guarantees against leaking of three types of private information from any agent to others.

Section 2 introduces background on JT-based organization and terminology. We show how private information is leaked by existing MAS frameworks in Section 3 and define measures to quantify such privacy loss. Section 4 reformulates hypertree construction into alternative problems that allow development of algorithms that fundamentally protect certain types of privacy. Section 5 presents an approach to construct a JT as a distributed maximum spanning tree, and shows the privacy loss by related methods. A novel algorithm DPMST is presented in Section 6 with the analysis of its soundness, complexity, and privacy. Section 7 considers hypertree existence recognition and proves two necessary and sufficient conditions. These conditions lead to a second algorithm HTBS for recognizing hypertree existence, that is presented in Section 8 with the analysis of soundness, by-product, complexity, and privacy. Experimental evaluations of DPMST and HTBS, as well as five alternative methods, are reported in Section 9. Proofs of formal results are collected at the Appendix.

2 Background and Terminology

2.1 Hypertree Based Frameworks

Decentralized probabilistic reasoning, constraint reasoning, and decision making are essential inference tasks of a MAS. We briefly overview inference frameworks that are based on hypertrees. These frameworks decompose an application environment into overlapping subenvironments with each agent controlling one. In the trouble-shooting environment, each subenvironment corresponds to an equipment component. Multiply sectioned Bayesian networks (MSBNs) [Xia96, Xia02] are the earliest framework for exact multiagent probabilistic reasoning based on hypertrees. The core of each agent is a Bayesian subnet over its subenvironment. Agents acquire local observations asynchronously. By passing two messages over each JT link, exact posterior probabilities relative to all observations are obtained at every agent. Typical applications include medical diagnosis [XPE$^+$93], equipment trouble-shooting [Xia02], and intelligent sensor networks [Xia08].

Multiple sectioned constraint networks (MSCNs) [XZ07, XMZ14] solve distributed constraint satisfaction problems with complex local problems such as university timetabling [XZ08]. The core of each agent is a constraint subnet that encodes variables and constraints over its subenvironment. The associated inference algorithm is sound and complete. DCTE [BM10] and Action-GDL [VRAC10] conduct distributed constraint optimization using JTs. They do not structure each subenvironment into a subnet. A major difference of MSCN from DCTE and Action-GDL is that each hypertree node in MSCN embeds a constraint subnet and inference is optimized within the subnet.

Collaborative decision networks (CDNs) solve multiagent decision problems such as collaborative design in supply chains [XCD04] and multiagent expedition [XH11]. The core of each agent is a decision subnet that encodes probabilistic dependency, alternative decisions, and preference relations. The optimal global design (of maximum expected utility) or team movement is computed distributively. CDN is also a hypertree-based framework for distributed constraint optimization. It differs from MSCN, DCTE and Action-GDL in that it models uncertainty with Bayesian probabilities and models objective functions as utility functions.

A subenvironment may be internally structured into a subnet as in MSBN, MSCN, and CDN, or unstructured as in Action-GDL and DCTE. Frameworks assuming structured subenvironments allow more efficient inference as subenvironments scale up. These subnets can conceptually be merged into a single global structure, which we refer to as the *dependency graph* of the MAS. In this paper, it suffices to assume that the dependency graph is an undirected graph (see below). For frameworks over unstructured subenvironments, we assume the existence of a subnet for each subenvironment where variables are pairwise connected.

2.2 Terminology

Let V be a collection of environment variables whose inter-dependency is described by a *dependency graph* $G = (V, E)$. Let $A = \{A_0, \ldots, A_{\eta-1}\}$ be a set of agents that decompose V into a set of overlapping *subenvironments* $\Omega = \{V_0, \ldots, V_{\eta-1}\}$, where $\cup_{i=0}^{\eta-1} V_i = V$ such that agent A_i controls subenvironment V_i. The tuple (G, A, Ω) specifies a MAS over the environment V.

If A_i and A_j ($j \neq i$) share variables, the intersection $I_{ij} = V_i \cap V_j \neq \emptyset$ is their *border* and the two agents are *bordering* or *adjacent*. Each variable in a border is a *shared* variable. A variable unique to a subenvironment is a *private* variable. For each agent A_i, the union of its borders $W_i = \cup_{j \neq i} I_{ij}$ is its *boundary*. The collection of agent boundaries $W = \{W_0, \ldots, W_{\eta-1}\}$ is the *boundary set* of the MAS.

The environment decomposition of a MAS can be visualized by an *environment decomposition graph*, a cluster graph where each cluster is a subenvironment and each link is a border. A boundary set can be visualized by a *communication graph*, a cluster graph where each cluster is a boundary and each link is a border. Fig. 1 illustrates a trivial MAS, whose dependency graph, environment decomposition graph, and communication graph are shown in (a), (b) and (c), respectively. The subenvironment of A_0 is $V_0 = \{a, b, u, y\}$, where a and b are private variables. The border between A_1 and A_2 is $I_{12} = \{y, z\}$. The boundary of A_1 is $W_1 = \{u, w, y, z\}$.

Our formulation of subenvironments differs from much of the constraint reasoning literature where

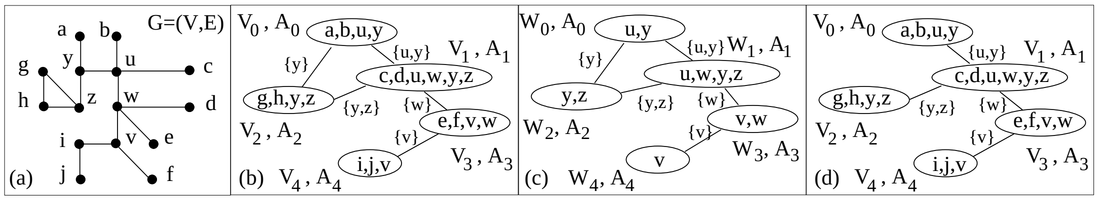

Figure 1: (a) Dependency graph. (b) Environment decomposition graph. (c) Communication graph. (d) JT organization.

variables are disjointly partitioned among agents that interact through shared constraints. For the border $\{y,z\}$ between A_1 and A_2, the alternative formulation may assign y to A_1 and z to A_2 with the agents sharing a constraint over y and z. As previously shown [XMZ14], the two formulations are equivalent. By sharing the constraint, both variables as well as their domains are known to both agents and are effectively shared. One exception [SSHF00] to the above does not reveal a constraint to all agents with variables in the constraint, although the method does not provide strong guarantee for non-disclosure.

In MAS literature, the number of variables controlled by an agent varies from one (e.g., Action-GDL) to hundreds or more. We focus on the MAS with complex local problems where each agent controls a large number of variables. It has been suggested [Yok01] that a complex local problem can be handled by a *virtual agent* that embeds multiple single-variable based *physical agents*. Under our formulation of subenvironments, no agent is truly single-variable. We will take the virtual agent perspective when discussing frameworks such as Action-GDL and refers to them as single-variable-per-physical-agent nevertheless.

A JT is a cluster tree where each link between two clusters is labeled by their nonempty intersection. The intersection of any two nonadjacent clusters is contained in every cluster on the path between the two (*running intersection*). Subenvironments (and agents) of a MAS may be organized into a JT, where each cluster is a subenvironment and each link is a border. Fig. 1 (d) shows a JT organization.

A JT organization prescribes direct message pathways between agents. Two agents can exchange messages whenever they are adjacent in the JT. Messages between adjacent agents during inference are restricted to concern shared variables only and two messages along each JT link (one in each direction) are sufficient to ensure sound inference. In multiagent probabilistic reasoning, a message encodes the sender agent's belief over shared variables, e.g., in MSBNs [Xia96, Xia02]. In multiagent constraint reasoning, a message encodes partial solutions over shared variables, e.g., in MSCNs [XZ07, XMZ14]. In multiagent decision making, a message is either an expected utility function over shared variables, or a partial action profile over these variables, e.g., in CDNs [XCD04, XH11].

2.3 JTs and Pseudotrees

While JTs are commonly used in probabilistic reasoning e.g., [Jen88, Xia02] and centralized constraint reasoning, e.g., [Dec03], pseudotrees are commonly used in multiagent constraint reasoning, e.g., [MSTY05, PF05]. For convenience of a broader readership, we briefly discuss their connection.

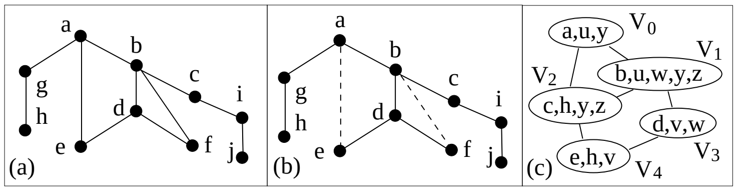

Figure 2: (a) A dependency graph. (b) A possible pseudotree of (a). (c) An environment decomposition graph (unrelated to (a)).

Given a dependency graph G, a pseudotree T with the same set of nodes and links can always be obtained. Start with any node, referred to as the *root*, traverse G depth first and backtrack when there is no adjacent unvisited node. Links traversed are *tree links* of T, directed away from the root, that define *parent/child* relations among nodes. Remaining links are *back links* of T, also directed away from the root, that define *pseudoparent/child* relations. Fig. 2 (b) shows a possible pseudotree obtained from (a) with the root a. An important property of a pseudotree is that nodes adjacent in G fall in the same branch of T.

An environment decomposition may not have a JT as it may be impossible to connect subenvironments into a tree with running intersection. Fig. 1 (b) has a JT in (d), but Fig. 2 (c) has no JT for the reason below. Consider clusters V_1, V_2, V_3, V_4 and possible links among them $\langle V_1, V_2\rangle, \langle V_1, V_3\rangle, \langle V_2, V_4\rangle, \langle V_3, V_4\rangle$. A JT cannot simultaneously contain all four links since a cycle is formed. If link $\langle V_1, V_2\rangle$ is absent, z is not running. If $\langle V_1, V_3\rangle$ is absent, w is not running. Similarly, running intersection cannot hold if any of the other two links is absent. Hence, Fig. 2 (c) has no JT.

3 Privacy for Agents Organized in Hypertree

3.1 Agent Privacy

As MAS organizations, hypertrees allow sound inference with the time complexity being linear on the number of agents [Xia02, XZ07, XMZ14, BM10, VRAC10, XH11]. As illustrated in the trouble-shooting example, cooperation requires information exchange but should respect privacy as much as possible. A hypertree has the potential for several types of privacy. Since messages during inference involve only variables shared between adjacent agents on the hypertree, at least three types of information need not be exchanged during inference and can potentially be kept private. We refer to them as three types of privacy.

Privacy on private variable This means non-disclosure of the identity (label) of any private variable and its domain (of possible values) to other agents. For instance, variable a of A_0 in Fig. 1 should not be revealed to A_1. Even if obfuscation is applied to variables and value labels, the disclosure still prompts targeted probing.

Privacy on shared variable This requires that the identity and domain of any shared variable should not be disclosed beyond agents that share it. For instance, variable w, shared by A_1 and A_3, should not be revealed to A_4.

Privacy on agent identity and bordering relation This requires non-disclosure of any agent's identity and its bordering relations to non-bordering agents. For instance, the identity of A_4 and its bordering relation with A_3 should not be revealed to A_1. We sometime refer to bordering relations as agent adjacency, which should not be confused with adjacency in a hypertree.

We refer to the three types collectively as agent privacy.

3.2 Privacy Loss in Hypertree Construction

A number of operations may be performed over the lifetime of a hypertree-based MAS. Among them, the most critical to agent privacy is the hypertree construction. Once the hypertree is constructed, inference naturally maintains privacy due to restriction on message content. It is through hypertree construction that agent privacy is compromised in existing frameworks, as we show below.

Action-GDL It is a single-variable-per-physical-agent framework [VRAC10], and we assume that a virtual agent embeds multiple physical-agents. For clarity, we equate nodes and variables to physical agents and reserve the word *agent* for a virtual agent. At the start, each variable in the dependency graph knows the identity of each adjacent variable and its domain.

A pseudotree of the dependency graph is first constructed by a distributed depth-first search (DFS). During DFS, each node only passes messages to adjacent nodes and no information on non-adjacent nodes is contained in messages. Hence, no privacy loss occurs during the pseudotree construction.

Next, the pseudotree is converted into a JT. Each pseudotree node forms its cluster by including (1) its own variable, (2) variables of its parent and pseudoparents, and (3) cluster variables of each child except the child variable. When a variable is shared by two agents, it is reasonable to assume that the corresponding pseudotree node is accessible by both agents. As the result, the JT cluster at the node is known to both agents. If the cluster includes a variable outside the subenvironment of one agent, privacy loss is incurred on both the identity and the domain of the variable as domain information is used by inference operation at the cluster.

Consider applying Action-GDL to environment decomposition in Fig. 3 (a). A possible pseudotree is shown in (b) and the resultant JT is shown in (c). Since variable b is contained in the subenvironment of A_0, A_0 has access to JT cluster $\{b,d\}$ at node b. Hence, private variable d of agent A_1 is leaked to A_0. Since inference at each cluster uses the domain of each variable in the cluster, the leak includes the variable domain. Similarly, d is also leaked to A_2. Furthermore, b is a variable shared only between A_0 and A_1. Through access to cluster $\{b,c,d\}$ at node c, b is leaked to A_2.

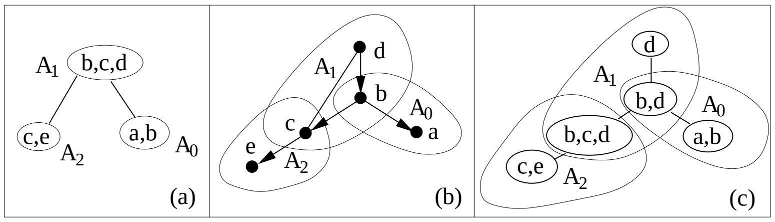

Figure 3: (a) Environment decomposition graph. (b) A pseudotree. (c) Resultant JT.

To summarize, Action-GDL incurs privacy loss on identity and domain for both private and shared variables, but has no loss on agent identity and adjacency.

DCTE The JT used by DCTE [BM10] is constructed by a method [PGM05] where each agent controls multiple variables. Two agents can exchange messages directly even if no variable is shared [PGM05]. Since this leads to unnecessary additional privacy loss, we consider a revised version where each agent only knows and sends messages to bordering agents.

First, the agents organize themselves into a spanning tree. In the process, a root is elected and a parent is selected for each agent. Each agent initializes its choice for the root and the parent to itself and sends the choice to bordering agents. By comparing agent IDs from messages, each agent updates its choice and passes on until an agent with the lowest ID is elected as the root of the spanning tree and every agent settles down on its parent in the spanning tree. Since the ID of the root choice can propagate far beyond adjacency, it causes privacy loss on agent identity. When agent A_i receives from A_j on its parent choice of A_k, adjacency of A_j and A_k is leaked to A_i.

Subsequently, each agent enlarges its subenvironment into a cluster that satisfies running intersection, which effectively transfers the spanning tree into a JT. If A_i is adjacent to A_j in the spanning tree, A_i sends to A_j its own variables and all variables reachable in the subtree rooted at A_i. If A_j receives two messages that both include variable x, it adds x to its cluster to ensure running intersection.

Fig. 4 (a) shows a subenvironment decomposition and (b) is a possible spanning tree. As shown in (c), messages during JT construction leak identities of all variables to each agent. Hence, this method has the maximum loss on both private and shared variable. To summarize, the JT construction by DCTE

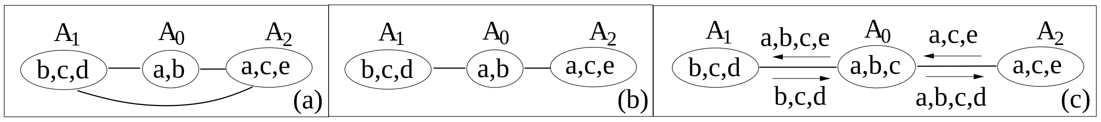

Figure 4: (a) Environment decomposition graph. (b) A spanning tree. (c) Resultant JT.

incurs all three types of privacy loss.

Both Action-GDL and DCTE are algorithm suits for distributed constraint optimization. Due to the focus of this work, their names in this paper refer to only the portion for hypertree construction.

COORD-Plus For MSBN, MSCN and CDN frameworks, a hypertree is constructed by a coordinator agent [Xia02, XMZ14] with the access of borders between each pair of agents, plus one private variable per agent (see Section 4.1). Assuming that the coordinator is one of the agents, it determines the existence of a hypertree, constructs one if exists, and informs each agent about the hypertree neighbors. We refer to this method as COORD-Plus. All agent identities and bordering relations, and identities of all shared variables are leaked to the coordinator, but there is nothing leaked to other agents. There is no loss on variable domain.

3.3 Quantification of Privacy Loss

We define a scheme to quantify the loss of the three types of privacy. Such quantification allows precise evaluation and comparison of privacy performance by alternative frameworks. VPS is one scheme [MPB$^+$06] whose quantification is based on estimates of an agent's possible state by other agents before and after an algorithm execution. It assumes that agent identities and their state spaces are publicly known. Although reasonable for applications such as meeting scheduling, this assumption does not hold in our context, as can be easily seen from the trouble shooting example. In the following, we develop an alternative scheme for quantifying the three types of privacy loss.

Let $border(A_i,A_j)$ be a function that returns 1 if A_i borders A_j and 0 otherwise. Let $know(A_i,A_j)$ be a function that returns 1 if A_i knows the identity of A_j and 0 otherwise. After a hypertree algorithm is run, the difference $know(A_i,A_j) - border(A_i,A_j)$ quantifies privacy loss on agent identity at A_i. Its value is 1 (one unit of loss) if A_i does not border A_j but the identity of A_j is leaked to A_i, and is 0 otherwise. The *system privacy loss* (SPL) on agent identity is

$$SPL_{aid} = \sum_i \sum_{j \neq i} (know(A_i,A_j) - border(A_i,A_j)).$$

The corresponding *maximum system privacy loss* (MSPL) is

$$MSPL_{aid} = \sum_i \sum_{j \neq i} (1 - border(A_i,A_j)).$$

For Fig. 1 (b), $MSPL_{aid} = 2+1+2+2+3 = 10$. The *normalized system privacy loss* (NSPL) on agent identity is

$$NSPL_{aid} = SPL_{aid}/MSPL_{aid} = \frac{\sum_i \sum_{j \neq i}(know(A_i,A_j) - border(A_i,A_j))}{\sum_i \sum_{j \neq i}(1 - border(A_i,A_j))} \in [0,1], \qquad (1)$$

where $NSPL = 1$ signifies the maximum loss and $NSPL = 0$ signifies no loss.

For privacy loss on bordering relation, let $knowBdr(A_i,A_j,A_k)$ be a function that returns 1 if A_j borders A_k and it is known by A_i, and returns 0 otherwise. The system privacy loss on bordering relation is

$$SPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} border(A_j,A_k) * knowBdr(A_i,A_j,A_k).$$

The corresponding maximum loss is

$$MSPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} border(A_j,A_k).$$

For Fig. 1 (b), $MSPL_{bdr} = 3+2+3+3+4 = 15$. The normalized system privacy loss on bordering relation is

$$NSPL_{bdr} = \frac{\sum_i \ \sum_{j \neq i} \ \sum_{k \neq i, k \neq j} border(A_j,A_k) * knowBdr(A_i,A_j,A_k)}{\sum_i \ \sum_{j \neq i} \ \sum_{k \neq i, k \neq j} border(A_j,A_k)} \in [0,1]. \qquad (2)$$

Let $private(x,A_i)$ be a function that returns 1 if variable $x \in V_i$ is private to A_i and 0 otherwise. Let $knowID(A_i,x)$ be a function that returns 1 if A_i knows the identity of x and 0 otherwise. The system privacy loss on private variable identity is

$$SPL_{pvid} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x,A_j) * knowID(A_i,x).$$

The corresponding maximum loss is

$$MSPL_{pvid} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j).$$

For Fig. 1 (b), $MSPL_{pvid} = 8+8+8+8+8 = 40$. The normalized system privacy loss on private variable identity is

$$NSPL_{pvid} = \frac{\sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j) * knowID(A_i, x)}{\sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j)} \in [0, 1]. \tag{3}$$

Let $knowDom(A_i, x)$ be a function that returns 1 if A_i knows the domain of x and 0 otherwise. The system privacy loss on private variable domain is

$$SPL_{pdom} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j) * knowDom(A_i, x).$$

The corresponding maximum loss is identical to $MSPL_{pvid}$ above. The normalized system privacy loss on private variable domain is

$$NSPL_{pdom} = \frac{\sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j) * knowDom(A_i, x)}{\sum_i \sum_{j \neq i} \sum_{x \in V_j} private(x, A_j)} \in [0, 1]. \tag{4}$$

Let $shared(x, A_i, A_j)$ be a function that returns 1 if $x \in V_i$ is shared by A_i and A_j, and 0 otherwise. The system privacy loss on shared variable identity is

$$SPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) * knowID(A_i, x).$$

The corresponding maximum loss is

$$MSPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k).$$

For Fig. 1 (b), $MSPL_{svid} = 4+2+4+5+6 = 21$. The normalized system privacy loss on shared variable identity is

$$NSPL_{svid} = \frac{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) * knowID(A_i, x)}{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k)} \in [0, 1]. \tag{5}$$

The system privacy loss on shared variable domain is

$$SPL_{sdom} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) * knowDom(A_i, x).$$

The corresponding maximum loss is identical to $MSPL_{svid}$ above. The normalized system privacy loss on shared variable domain is

$$NSPL_{sdom} = \frac{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) * knowDom(A_i, x)}{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k)} \in [0, 1]. \tag{6}$$

Note that privacy loss on variable identity and loss on variable domain are independent in general. For instance, COORD-Plus leaks identities of shared variables to the coordinator, but the domains of these variables are not leaked.

The problem of privacy preserving hypertree construction can be expressed with the above measures.

Problem 1 (Privacy preserving hypertree construction) *Given the environment decomposition of a MAS such that a hypertree exists, how can agents construct a hypertree, while keeping privacy loss as small as possible as measured by Eqns. (1) through (6)?*

4 Problem Reformulations

4.1 Boundary Set Based Problem Reformulation

To solve Problem 1, we reformulate it as follows. It was observed [Xia02] that private variables of a subenvironment can be aggregated into one. Hence, after COORD-Plus constructed JT using the aggregated subenvironments, each aggregated private variable is replaced with the original set of private variables. The new cluster tree is a valid JT. We claim that the aggregated private variable can be avoided, which leads to a more succinct problem context. Proposition 1 establishes that a JT-based organization can be constructed without referencing private variables.

Proposition 1 *Given V, Ω, and W of a MAS, let T be a JT with boundaries in W as clusters. Let T' be a cluster tree with subenvironments in Ω as clusters, such that it is isomorphic to T with each subenvironment mapped to the corresponding boundary in T. Then T' is a JT.*

We refer to the JT T in Proposition 1 as a *boundary based JT*. Based on Proposition 1, we reformulate Problem 1.

Problem Reformulation 1 (Boundary set based hypertree construction) *Given the boundary set of a MAS such that a boundary based JT exists, how can agents construct such a JT, while keeping the privacy loss as small as possible as measured by Eqns. (1), (2), (5), and (6)?*

Eqns. (3) and (4) are left out from the reformulation, as the loss due to private variables is guaranteed to be zero. The reformulated context can be elaborated as follows.

- A JT exists with boundaries in W as clusters.
- For every pair of $i \neq j$, A_i and A_j know each other and can pass messages if they have a border $I_{ij} = W_i \cap W_j \neq \emptyset$.
- For each i, A_i knows nothing about variables in other boundaries and outside W_i.

To solve the reformulated Problem 1, the task of agents is to compute a boundary based JT such that each agent knows its adjacent agents in the JT, and the process does not disclose information on agent identity, border relation and boundary beyond the above knowledge state.

4.2 Maximum Spanning Tree Based Problem Reformulation

To solve the above problem, we explore the relation between JT and maximum spanning tree (MST).

1. Given a boundary set W with a JT, let CG be the communication graph and Ψ be a weighted graph isomorphic to CG. That is, for each boundary $W_i \in W$, create a node x_i in Ψ. Add a link $\langle x_i, x_j \rangle$ to Ψ if there is a border I_{ij} between W_i and W_j. Assign the link weight as $w(x_i, x_j) = |I_{ij}|$. Fig. 5 (a) and (b) illustrate CG and Ψ.
2. Let Ψ' be any MST of Ψ and T be a cluster tree subgraph of CG isomorphic to Ψ'. That is, T has the same set of clusters as CG and, for each link $\langle x_i, x_j \rangle$ of Ψ', W_i and W_j are adjacent in T. Fig. 5 (c) and (d) illustrate Ψ' and T.

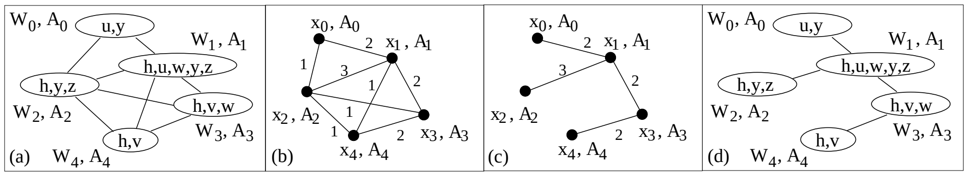

Figure 5: (a) A communication graph. (b) The weighted graph isomorphic to (a). (c) A MST of (b). (d) The cluster tree subgraph of (a) isomorphic to (c).

Proposition 2 asserts the nature of T. It is based on a well-known result, e.g., [Jen88, Xia02], phrased in the current context. It follows from Proposition 2 that Fig. 5 (d) is a JT that solves the reformulated Problem 1.

Proposition 2 *The cluster tree T is a JT, iff a boundary based JT exists.*

In the above process, the communication graph CG and weighted graph Ψ can both be specified from local information on W at each agent. Once the MST Ψ' is obtained, the cluster tree T can be specified from local information on Ψ' and W. Hence, there is no disclosure of privacy in these steps. The critical step is constructing MST Ψ' from Ψ. This leads to another equivalent reformulation of Problem 1, based on Proposition 2.

Problem Reformulation 2 (MST based hypertree construction) *Given the boundary set of a MAS such that a boundary based JT exists, let Ψ be a weighted graph isomorphic to the communication graph. How can agents construct a MST from Ψ, while keeping privacy loss as small as possible as measured by Eqns. (1) and (2)?*

Eqns. (5) and (6) are left out from the reformulation, as only a count of shared variables between bordering agents is used, with their identities and domains excluded. The context of the reformulated problem can be elaborated as follows.

- Agents are one-to-one mapped to nodes in the weighted graph Ψ. Hence, agent identity and node identity are interchangeable below, and so are agent bordering relation and node adjacency.
- For every pair of $i \neq j$, A_i and A_j know each other and the weight $w(x_i, x_j)$, and can pass messages, if they are adjacent in Ψ.

To solve the reformulated problem, the task of agents is to compute a MST such that each node knows adjacent nodes in the MST, and the process minimizes disclosure of node identity and adjacency.

5 Distributed MST Construction

5.1 Work Related to Distributed MST Construction

Before presenting our distributed MST algorithm, we review the relevant literature. Among them, we refer to the pioneering work [GHS83] as GHS and describe it in details. Since algorithms for minimum or maximum spanning trees differ only in the comparison operator (Min versus Max), we refer to both as the MST.

GHS The method is based on the notion of MST fragments, each initially made of a single node. Smaller fragments are combined into larger ones concurrently based on a level control until a single fragment (the MST) is left. It has a time complexity $O(\eta\ log\ \eta)$. Each fragment of more than one node is identified by the weight of a core link. This fragment identity is propagated to all nodes of the fragment to coordinate growth. To ensure unique fragment identities, GHS assumes distinct link weights, which generally does not hold. To accommodate nondistinct weights, the weight of each link is appended with identities of its end nodes. Hence, a fragment identity contains node identity and adjacency.

To summarize, when GHS is applied to MST based hypertree construction, each node corresponds to an agent. Hence, it incurs privacy loss on agent identity and adjacency through the propagation of fragment identities. Due to the problem reformation, there is no loss on private and shared variables.

Other methods Awerbuch [Awe87] proposed a three-stage algorithm, which was later improved [FM95], with time complexity $O(\eta)$. It starts with a counting stage to get η. Then GHS is run to grow each fragment to an $\Omega(\eta/log\ \eta)$ size. A variant of GHS follows, with a more accurate level updating to speed up computation. Non-distinct link weights are handled using the same technique as GHS and hence the method suffers the same privacy loss.

An improved algorithm [GKP98] is proposed with time complexity $O(d + \eta^{0.613}\ log*\ \eta)$, where d is the diameter (maximum length of a simple path) of the weighted graph. It first uses a variant of GHS to produce multiple fragments of small diameters and then combines them into a MST by a rooted operation. Its limitation on privacy is identical to GHS. Another algorithm [KP98] consists of two parts. In the first part, a $\sqrt{\eta}$-dominating set D of size at most $\sqrt{\eta}$ is computed as well as a partition of the weighted graph into fragments, one per node in D. The second part combines the fragments into a MST by the rooted operation above [GKP98]. Since the first part employs a simplified GHS, its limitation on privacy is identical.

One algorithm [KP08] computes an approximate MST. Due to the necessary and sufficient relation between JT and MST (Proposition 2), an approximate MST cannot yield a JT. Hence, the method is not applicable to the reformulated Problem 1. Another algorithm [NCKB12] computes a set of MSTs, one for each component of a disconnected graph. As a parallel algorithm, access of the entire graph by each processor is assumed. Hence it is applicable only when privacy is not a concern.

In summary, applicable existing methods of distributed MSTs all incur privacy loss on agent identity and adjacency when applied to MST based hypertree construction. In Section 6, we present *DPMST* as a solution to Problem 1 that incurs no loss on agent identity and significantly less loss than GHS on agent adjacency.

5.2 Distributed MST for Privacy on Agent Identity and Bordering Relation

We introduce the main idea of *DPMST* here and specify it formally in Section 6. For improved privacy, we take a different direction from GHS and its extensions [Awe87, FM95, GKP98, KP98]. Rather than growing multiple fragments simultaneously, we extend Prim's algorithm [Pri57] distributively and grow a MST through a rooted control. As the result, *DPMST* does not assume distinct link weights and needs not append node identities to link weights.

Precisely stated, the task is as follows. Given a distributed representation of a connected, weighted graph Ψ of η nodes, construct a MST Ψ' by distributed computation. For adjacent nodes v and x, the weight of link $\langle v,x\rangle$ is $w(v,x)$. Each node initially knows each adjacent node (*neighbor*) and their link weight. It knows nothing about other nodes and links between them.

DPMST initializes Ψ' with a node in Ψ, referred to as the *root*, and builds Ψ' up as a *directed* single-rooted tree in $\eta - 1$ rounds. An *outgoing* link of Ψ' is a link of Ψ with only one end in Ψ'. In each round, a best outgoing link $\langle p,c\rangle$ (with a maximum weight) is selected, where p is in Ψ', and both c and $\langle p,c\rangle$ are added to Ψ'. We refer to p as the *tree-parent* of c and c as a *tree-child* of p. For any node in Ψ', we refer to its tree-parent or a tree-child as its *tree-neighbor*. Each node v maintains the following data.

1. The state of v is indicated by a variable $state \in \{OUT, IN, DONE\}$. OUT means that v is not in Ψ', IN means that v is in Ψ', and $DONE$ means that v is terminated (halted).
2. The state of each neighbor x is maintained by a variable $nbstate(x) \in \{OUT, IN, DONE\}$.
3. A pointer *tree-parent* points to the parent of v in Ψ'.
4. A *weight table* is maintained after v is added to Ψ'. Each row is indexed by a neighbor x of v, that either corresponds to an outgoing link (if x is not in Ψ'), or leads to an outgoing link (if x is in Ψ'). A weight denoted by $w(x)$ is stored at each row. If x is not in Ψ', $w(x) = w(v,x)$. Otherwise, $w(x)$ is the weight of the best outgoing link through x in Ψ'.

During execution, four types of messages are passed.

Notify A MST tree-leaf notifies a neighbor that the latter is added to the current MST.

Announce The sender announces to each neighbor that the former is in the MST.

Expand The sender instructs a tree-child to expand the current MST by adding a new node.

Report The sender reports to its tree-parent either the best outgoing weight (*bow*) or termination.

These messages carry no information on node identity or adjacency. We make the standard assumptions on message transmission: (1) Transmission of each message takes at most one time unit; (2) Messages are received in order of sending; (3) Local computation time at each node can be ignored. The example below illustrates how *DPMST* uses the above messages. It is formally specified in Section 6.

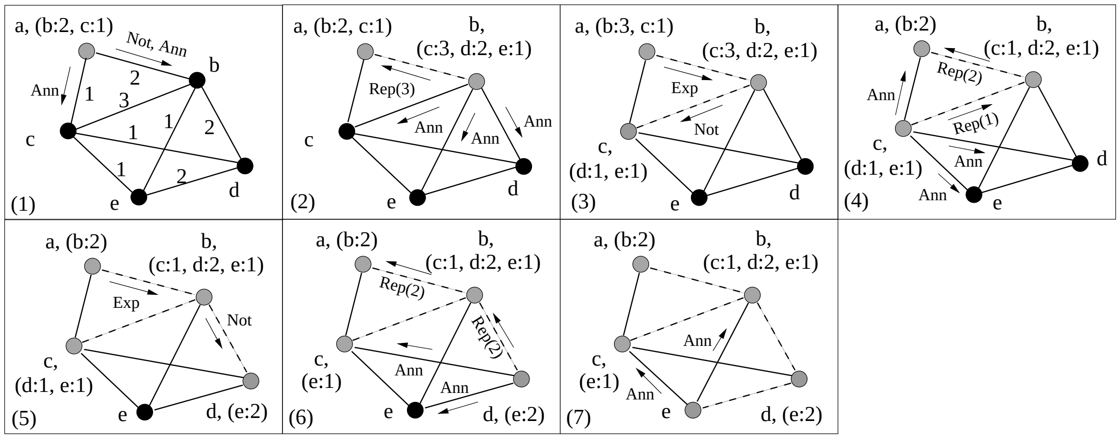

Figure 6: Illustration of an execution of *DPMST*

Example 1 *Consider the weighted graph in Fig. 5 (b). For simplicity, we refer to nodes x_0 through x_4 as a, b, c, d, e, shown in Fig. 6 (1). Suppose a is the root, whose weight table is created as $(b:2;\ c:1)$. It sends Announce to b and c, and $Notify$ to b. In response, b creates its table as $(c:3;\ d:2;\ e:1)$, sends Announce to c, d and e, and Report with bow $= 3$ to a. Ψ' now contains nodes a, b (grey) and link $\langle a, b\rangle$ (dashed) as shown in (2). Although the Announce from a to b can be saved, we omit such optimization for simplicity.*

Based on the Report, node a revises its table to $(b:3;\ c:1)$ and sends $Expand$ to b. To expand, b sends $Notify$ to c, which in turn creates its table as $(d:1;\ e:1)$. Ψ' now contains a, b and c as in (3).

Node c sends Announce to a, d and e, and Report to b with bow $= 1$. When node a receives Announce from c, it revises its table to $(b:3)$. When b receives Report from c, it revises its table to $(c:1;\ d:2;\ e:1)$, and sends Report to a with bow $= 2$. Upon receiving the Report, a revises its table to $(b:2)$ as in (4).

Next, node a sends $Expand$ to b, which in turn sends $Notify$ to d. Node d creates its table as $(e:2)$. Ψ' now contains a, b, c and d, shown in (5). Node d sends Announce to c and e, and Report to b with bow $= 2$. In response, c revises its table to $(e:1)$, and b reports to a, shown in (6).

In the final round, $Expand$ propagates from a to b and then to d, with d sending $Notify$ to e. Node e sends Announce to b and c. Since it has no OUT neighbors, e sends Report to d on its termination. Its Announce to c causes c to terminate and its Report to d causes d to halt. Before halting, both c and d send Report to b, together with the Announce from e, causing b to halt. The Report that b sends to a before halting causes a to terminate. Now DPMST is completed, with Ψ' shown by the dashed links in Fig. 6 (7). It is identical to Fig. 5 (c). The messages contain no agent identity or adjacency.

The weight table at each node plays a critical role. Using the table, the root decides the neighbor to $Notify$ or the tree-child for $Expand$. Each non-root tree-parent instructed to $Expand$ does the same. Hence, correct weight table updating is critical. An updating may be triggered by adding a new MST node x. After a sequence of $Expand$ and $Notify$ reaches x, it reports its best outgoing weight based on the outgoing links. The reports propagate along the $Expand/Notify$ path back to the root and each node on the path updates its table. A table updating may also be triggered by *Announce* sent from x. Suppose a neighbor y of x is in the MST before x. Then the table at y maintains $w(x)$. Upon receiving *Announce* from x, x is no longer outgoing from y and $w(x)$ is deleted. The deletion may reduce the best outgoing weight from y or cause y to terminate. In either case, the reports propagate along the tree-ancestor path of y to the root.

If x has r neighbors, up to r sequences of reports, one along the $Expand/Notify$ path from x to the root and each other along a tree-ancestor path from y to the root, race to the root. If the root makes the $Expand$ decision upon arrival of the first report, its table may not be updated correctly, yielding a wrong decision. On the other hand, how many *Announce*-induced reports will ever arrive is generally unknown to the root. Hence, waiting of k reports for some finite k is not optional.

We develop a rule by which the root waits for a maximum amount of time to ensure all relevant reports have arrived. First, the root keeps track the maximum length of $Expand/Notify$ pathes. When a $Notify$ message reaches a new MST node x, it reports to the tree-parent with a parameter $out\_hop = 1$. When the *Report* is relayed towards the root, each MST node sets $out\_hop = in\_hop + 1$. When the root receives the *Report*, its in_hop parameter is the length h of the MST path from x. The root maintains $\delta = \max_h(h)$, maximized over all h reported. δ is the radius of the current MST centered at the root.

Next, we analyze the timing for the *Announce*-induced *Report*. Suppose the root sends out *Expand* at t_0, x sends the first *Announce* at t_1, the root receives the $Notify$-induced *Report* at t_2, a MST node y receives an *Announce* at t_3, and the root receives the *Announce*-induced *Report* at t_4. The following equations hold where time is measured by the number of message transmissions. The transmission of h $Expand/Notify$ messages occurs during period $[t_0, t_1]$:

$$t_1 - t_0 = h. \tag{7}$$

Assuming that x sends all *Announce* messages before its *Report*, the transmission of $r-1$ *Announce* messages and h $Notify$-induced *Report* messages occurs in $[t_1, t_2]$, where r is the degree of x:

$$t_2 - t_1 = r - 1 + h. \tag{8}$$

Period $[t_1, t_4]$ is bounded by the time to transmit $r-1$ *Announce* messages and δ *Announce*-induced *Report* messages:

$$t_4 - t_1 \leq r - 1 + \delta. \tag{9}$$

From the above two equations, we derive

$$t_4 - t_2 \leq \delta - h. \tag{10}$$

$\delta - h$ is the maximum time that the root waits after receiving the $Notify$-induced *Report* to ensure all *Announce*-induced *Report*s are arrived. Proposition 3 estimates the necessary wait time of the root by refining the above analysis. It is used later in a complexity analysis.

Proposition 3 *Let the MST root send an Expand request at t_0 and receive the $Notify$-induced Report with parameter h. Let δ be the current radius parameter at the root and r be the degree of the new MST node. Then, all Announce-induced Reports arrive no later than $t = t_0 + r - 1 + h + \delta$.*

6 *DPMST* Algorithm Suite

6.1 The Algorithm Suite

We specify the *DPMST* algorithm suite, as a set of event-driven procedures. Before responding to messages, every node in the weighted graph Ψ is initialized so that $state = OUT$, $tree - parent = nil$, and $nbstate(x) = OUT$ for each neighbor x.

An arbitrary node z acts as the *root* that starts the process by executing $Start$. It adds itself to the empty MST Ψ', creates the weight table, and runs *Expand* to expand Ψ'. We assume that the root is arbitrarily specified externally, as its choice does not affect correctness of the outcome. It corresponds naturally with reality that hypertree construction is one of the first operations of a MAS.

Procedure 1 (Start)

```
1  $state = IN$; $\delta = 0$;
2  create weight table with one row per neighbor;
3  for each row of table indexed by $x$, $w(x) = w(z,x)$;
4  send Announce message to each neighbor;
5  run Expand;
```

Node v that runs *Expand* may be the root to continue $Start$ (as above) or to respond to a timer expiration (see Proc. 6). If v is not the root, it must be in Ψ' to respond to *Expand* from its tree-parent. Using the weight table, v selects a neighbor y that leads to a best outgoing link. It adds y to Ψ' if y is OUT. Otherwise, it asks y to expand.

Procedure 2 (Expand)

```
select neighbor $y = \arg\max_x w(x)$ from weight table, breaking ties randomly;
if $nbstate(y) = OUT$,
    send $Notify$ message to $y$;
    $nbstate(y) = IN$; record $y$ as a tree-child;
else send $Expand$ message to $y$; // $y$ must be $IN$
```

When node v receives $Notify$ from neighbor p, it is added to Ψ' and runs Proc. 3 in response. In the process, it creates the weight table, announces the new status to neighbors, and sends *Report* to p by Proc. 4.

Procedure 3 (Response to Notify)

```
state = IN; nbstate(p) = IN; tree-parent = p;
create weight table with one row per neighbor $x$, where $nbstate(x) = OUT$;
for each row of table indexed by $x$, $w(x) = w(v,x)$;
for each neighbor $y \neq p$, send Announce message to $y$;
run Inform(N, hop=1);
```

A node v runs $Inform$ to send *Report* to its tree-parent. The message has several arguments. *NA* indicates whether the *Report* is $Notify$-induced ($NA = N$) or *Announce*-induced ($NA = A$). Argument *hop* is only used in when $NA = N$. Argument $state$ signifies termination of the sub-MST rooted at v. The best outgoing weight *bow* from the weight table influences the next round of expansion.

Procedure 4 (Inform(NA, hop))

```
if there is no neighbor $y$ with $nbstate(y) = OUT$ and each tree-child $c$ has $nbstate(c) = DONE$,
    if $v$ is not root, send message $Report(NA, hop, state = DONE, bow = nil)$ to tree-parent;
    $state = DONE$; halt;
else if $v$ is not root,
    compute $maxbow = \max_x w(x)$ from weight table;
    send message $Report(NA, hop, state = nil, bow = maxbow)$ to tree-parent;
```

When node v receives *Announce* from neighbor x, it performs the following. The root cannot receive *Announce* from a tree-child, but can receive from a non-child tree-descendent.

Procedure 5 (Response to Announce)

```
$nbstate(x) = IN$;
if weight table has been created, delete its row indexed by $x$;
if $state = IN$, run Inform(A, hop=nil);
```

When note v receives *Report* from a tree-child c, it performs Proc. 6. It updates the weight table and sends *Report* to the tree-parent. If v is the root and *Report* is $Notify$-induced, v sets an expire time and updates the radius. When the timer expires, it starts the next $Expand$.

Procedure 6 (Response to Report(NA, hop, state, bow))

```
if $NA = N$, hop++;
if argument $state = DONE$,
    $nbstate(c) = DONE$; delete the row indexed by $c$ from weight table;
    if there is no neighbor $y$ with $nbstate(y) = OUT$ and each tree-child $c$ has $nbstate(c) = DONE$,
        if $v$ is not root, send $Report(NA, hop, state = DONE, bow = nil)$ to tree-parent;
        $state = DONE$; halt;
else if argument $bow \neq w(c)$ in weight table, $w(c) = bow$;
if $v$ is not root,
    compute $maxbow = \max_x w(x)$ from weight table;
    send $Report(NA, hop, state = nil, bow = maxbow)$ to tree-parent;
else if $NA = N$, // $v$ is root
    set timer to $\max(\delta - hop, 0)$; $\delta = \max(\delta, hop)$;
```

The timer setting at line 12 is based on Eqn. (10). From Procs. 4 and 6, when a node changes its state to *DONE*, it is terminated. The behavior of the algorithm suite is illustrated in Example 1.

6.2 Soundness, Complexity, and Agent Privacy

The soundness of *DPMST* is established in Proposition 4.

Proposition 4 *Given a connected, weighted graph Ψ, DPMST computes a MST Ψ' of Ψ specified distributively such that each node knows its tree-neighbors.*

Proof: *Announce* messages to neighbors and recursive reporting in Procs. 4 and 6 let each current MST node know through which neighbor the best outgoing link in its sub-MST can be reached. Based on Eqn. (10), by communicating *NA* and *hop* count and controlling wait time, potential incorrect expansion decisions due to transient weight tables are avoided. Recursive *Expand* messages combined with *Notify* add the best outgoing link to Ψ' in each round. Therefore, *DPMST* extends Prim's algorithm distributively and computes a MST correctly.

When a node v is added to Ψ', it knows its notifier as the tree-parent, and its notifier knows v as a tree-child. Hence, when *DPMST* halts, each node knows its tree-neighbors in Ψ'. □

We analyze communication cost and time complexity below. Let d denote the *diameter* of Ψ, e denote the number of links, and r denote the maximum degree of nodes. For communication cost, each node is added to Ψ' with at most d *Notify*/*Expand* messages: a subtotal of $O(d\ \eta)$ messages. Each link of Ψ passes two *Announce* messages, one for each end when added to Ψ': a subtotal of $O(2e)$ messages. After a node is added to Ψ', *Report*s are propagated to the root from the node (Proc. 3) and its neighbors (Proc. 5): $O(r\ d)$ messages. This yields a subtotal of $O(r\ d\ \eta)$ messages. Hence, the total number of messages is $O(r\ d\ \eta + 2e)$.

For time complexity, by Proposition 3, each round of expansion is bounded at $O(2d+r)$ time. A total of $O(\eta)$ rounds has the time complexity $O((2d+r)\ \eta)$.

For agent privacy, consider the overall context of solving Problem 1. By using the boundary set based problem reformulation, there is no loss on private variable. By using the MST based problem reformulation, there is no loss on shared variable. During *DPMST*, messages are exchanged between neighbors only and contain no node identity or adjacency. Hence, there is no loss on agent identity. We consider below whether *DPMST* may disclose some agent bordering relations.

Example 2 *Apply DPMST to Fig. 7 (1), where a is the root and the MST currently contains link $\langle a,b\rangle$ only. Next, a asks b to expand and b notifies e. In response, e announces to a and d, and reports to b. By relating the Expand to b, the Report with $hop = 2$, and the Announce from e, node a infers the adjacency $\langle b,e\rangle$.*

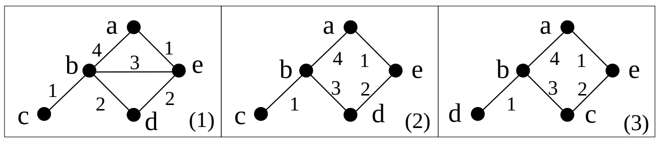

Figure 7: Three weighted graphs

The inference can also occur at a MST node other than the root, as summarized by the rule below.

Rule to infer adjacency Let x be the tree-parent of y. After x sends an *Expand* to y, if x receives *Announce* from neighbor z and *Report* from y with $hop = 2$, x can infer the adjacency between y and z.

Such inference is possible only within the immediate neighborhood of a node. The example below shows that reliable adjacency inference is generally impossible.

Example 3 *Apply DPMST to Figs. 7 (2) and (3), where a is the root and the MST currently contains link $\langle a, b\rangle$ only. In both cases, the following sequence of events occur relative to a. Node a asks b to expand and later b reports to a with $hop = 2$, where d (in (2)) or c (in (3)) is added to the MST. Node a asks b to expand and as the result e is added to the MST. What a perceives is Announce from e and Report from b with $hop = 3$. Subsequently, a asks b to expand and later b reports to a with $hop = 2$, due to addition of c (in (2)) or d (in (3)) to the MST. Since the same sequence of events occur in both cases, a cannot tell which weighted graph it is in. This inability primarily originates from non-existence of node identity in DPMST messages.*

In Section 9, we report the amount of privacy loss by DPMST due to the above rule.

7 Necessary and Sufficient Conditions for Hypertree Existence

7.1 Hypertree Existence

We have presented how to construct a hypertree from a given environment decomposition Ω, assuming that a hypertree exists. In general, a given Ω may not have a hypertree. For example, Fig. 2 (c) has no JT. Hypertree existence is recognized centrally by COORD-Plus [Xia02], that incurs privacy loss on agent identity, bordering relation, and shared variable. In this and next section, we address distributed, privacy preserving recognition of hypertree existence.

Problem 2 (Privacy preserving recognition of hypertree existence) *Given the environment decomposition of a MAS, how can agents determine whether a hypertree exists, while keeping privacy loss as small as possible as measured by Eqns. (1) through (6)?*

Similarly to Proposition 1 that shows that hypertrees can be built without reference to private variables, Proposition 5 says that hypertree existence can be determined without such reference.

Proposition 5 *Given Ω and W of a MAS, if there exists no JT with boundaries in W as clusters, then there exists no JT with subenvironments in Ω as clusters.*

Based on Proposition 5, we reformulate Problem 2.

Problem Reformulation 3 (Boundary set based recognition of hypertree existence) *Given the boundary set of a MAS, how can agents determine whether a hypertree exists, while keeping privacy loss as small as possible as measured by Eqns. (1), (2), (5), and (6)?*

Eqns. (3) and (4) are left out from the reformulation, as zero loss on private variable is guaranteed.

7.2 Boundary Graph Based Condition

Proposition 5 establishes whether a hypertree exists soly depends on the boundary set. However, it gives no guidance on how to recognize the existence. We identify a necessary and sufficient condition that provides such guidance. It is described through an alternative representation of the boundary set W. An undirected graph is the *boundary graph* of a MAS, if the set of nodes is $N = \cup_{i=0}^{\eta-1} W_i$ and each W_i is complete (elements pairwise connected). Fig. 8 (a) reproduces the boundary set in Fig. 1 (c) with the boundary graph in Fig. 8 (b).

Lemma 1 establishes a condition under which a JT can be constructed from a boundary graph, where each cluster is a boundary (although not every boundary is a cluster). A set of nodes in a graph is a *clique* if they are maximally pairwise connected. Two clusters are *comparable* if one is a subset of the other.

Lemma 1 *Let W be the boundary set of a MAS and BG be its boundary graph such that*

1. *BG is chordal and*
2. *for each clique C in BG, there exists $W_i \in W$ with $C \subseteq W_i$.*

Let T be a JT whose clusters are cliques in BG. For every cluster Q in T, there exists a boundary $W_i = Q$.

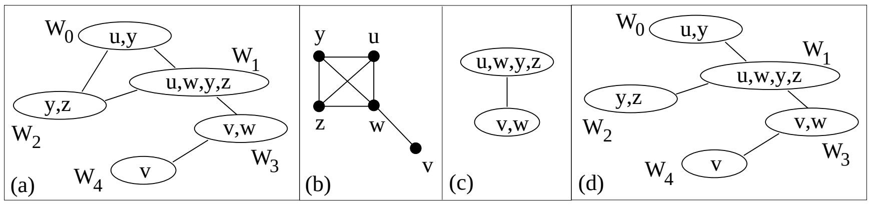

Figure 8: (a) Communication graph. (b) Boundary graph. (c) JT from (b). (d) JT of (a).

Example 4 *We illustrate Lemma 1 by Fig. 8. The boundary graph in (b) is chordal. It has two cliques $\{u,w,y,z\}$ and $\{v,w\}$. Each is contained in a boundary in (a), i.e., W_1 and W_3, respectively. Hence, both subconditions of Lemma 1 hold. The JT T stated in the lemma is shown in (c).*

Lemma 1 says that every cluster in T is a boundary. The inverse does not hold in general. That is, not every boundary is a cluster in T, e.g., W_0.

Using Lemma 1, Theorem 1 provides a necessary and sufficient condition for hypertree existence.

Theorem 1 *Let W be a boundary set with the boundary graph BG. A JT exists iff the following holds.*

1. *BG is chordal and*
2. *for each clique C of BG, there exists a boundary $W_i \in W$ such that $C \subseteq W_i$.*

Theorem 1 implies that, as far as hypertree existence is concerned, a boundary set falls into one of three mutually exclusive and exhaustive types.

Type 1 Boundary graphs are chordal and their cliques are boundary contained.

Type 2 Boundary graphs are not chordal.

Type 3 Boundary graphs are chordal but their cliques are not boundary contained.

Example 5 *Consider the boundary set in Fig. 8 (a) whose boundary graph is in (b). The boundary set is Type 1. The JT from the boundary graph is in (c). The two clusters are associated with agents A_1 and A_3. After adding a cluster for each of the three remaining agents, the boundary based JT is shown in (d).*

Example 6 *Fig. 9 (a) shows an environment decomposition with the boundary set in (b) and BG in (c). Since BG is not chordal, the boundary set is Type 2. By Theorem 1, it has no JTs.*

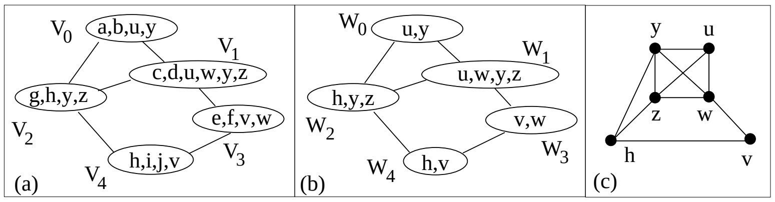

Figure 9: (a) Environment decomposition graph. (b) Communication graph. (c) Boundary graph.

Example 7 *For the boundary set in Fig. 10 (a), BG in (b) is chordal with two cliques. Since one of them $\{h,v,w\}$ is not contained in any boundary, the boundary set is Type 3. By Theorem 1, it has no JTs.*

Typing the boundary set guides algorithm development below for recognition of JT existence.

7.3 Boundary Set Based Condition

We identify a second necessary and sufficient condition for hypertree existence, which directly leads to distributed existence recognition. First, we define an operation to eliminate a boundary from the boundary set W, relative to a bordering boundary. When a boundary $W_i \in W$ is *eliminated relative to* a

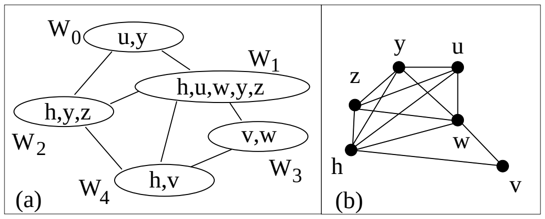

Figure 10: (a) Communication graph. (b) Boundary graph.

boundary W_j where $i \neq j$ and $W_i \cap W_j \neq \emptyset$, it yields a *reduced boundary set* $W' = (W \setminus \{W_i, W_j\}) \cup \{W'_j\}$, where

$$W'_j = \bigcup_{k \neq i, k \neq j} (W_j \cap W_k).$$

That is, the set W' resultant from eliminating W_i relative to W_j is obtained by deleting W_i and W_j from W, and replacing with W'_j. W'_j is obtained by the union of borders of A_j, except the border with A_i. In other words, W'_j is the boundary W_j without variables that A_j uniquely shares with A_i.

Example 8 *Consider the boundary set in Fig. 8 (a), $W = \{W_0, ..., W_4\}$. After W_0 is eliminated relative to W_1, the* reduced boundary set *is $W' = \{W'_1, W_2, W_3, W_4\}$, where $W'_1 = \{w, y, z\}$.*

Without confusion, we refer to each element of W' as an *active boundary*, whether or not it is an element of W. The elimination operation is well defined on the reduced boundary set and hence can be applied iteratively. In the case where only two identical active boundaries are left, i.e., boundary set becomes $\{W_i, W_j\}$ with $W_i = W_j$, we define the replacement $W'_j = \emptyset$. That is, $W' = \{\emptyset\}$.

Example 9 *For the boundary set in Fig. 8 (a),*

$$W = \{W_0 = \{u, y\}, W_1 = \{u, w, y, z\}, W_2 = \{y, z\}, W_3 = \{v, w\}, W_4 = \{v\}\},$$

eliminations can be performed iteratively as follows.

$$\begin{array}{lll}
Eliminate\ \{u, y\} & relative\ to\ \{u, w, y, z\}: & W' = \{\{w, y, z\}, \{y, z\}, \{v, w\}, \{v\}\}; \\
Eliminate\ \{v\} & relative\ to\ \{v, w\}: & W' = \{\{w, y, z\}, \{y, z\}, \{w\}\}; \\
Eliminate\ \{w\} & relative\ to\ \{w, y, z\}: & W' = \{\{y, z\}, \{y, z\}\}; \\
Eliminate\ \{y, z\} & relative\ to\ \{y, z\}: & W' = \{\emptyset\}.
\end{array}$$

Each W_i eliminated relative to W_j has been chosen to satisfy $W_i \subseteq W_j$. Its significance is seen below. Each reduced set W' (except the final singleton) is a well-defined boundary set in the sense that each variable is shared by at least two boundaries in W'. Take $W' = \{\{w, y, z\}, \{y, z\}, \{w\}\}$ for example. Each of w, y, and z is shared by two boundaries.

Next, we establish the second necessary and sufficient condition based on boundary elimination.

Theorem 2 *A MAS with the boundary set W has a JT agent organization, iff W can be eliminated iteratively into a singleton such that each W_i eliminated relative to W_j satisfies $W_i \subseteq W_j$.*

Theorem 2 suggests the following idea for a distributed algorithm to recognize hypertree existence.

7.4 Distributed Recognition of Hypertree Existence

Condition $W_i \subseteq W_j$ in Theorem 2 is equivalent to $W_i = I_{ij}$ and we refer to as *border equality*. Since it is a local condition, a privacy preserving, distributed recognition can proceed as follows. Agents are self-eliminated one by one as long as possible. A_i can be eliminated if border equality holds relative to a remaining agent A_j. After elimination, A_j removes from its boundary variables shared uniquely with A_i. If all agents are eliminated except one (whose active boundary becomes $\emptyset$), then a hypertree exists.

A token is passed between bordering agents by depth-first-traversal (DFT). The first round of DFT starts at an agent with token tok^1. If an agent A_i holds the token and satisfies border equality relative to another agent A_j, then A_i signifies to each bordering agent that it is eliminated and passes a new token tok^2 to A_j. A_j then starts the second round of DFT with remaining agents using tok^2. If an agent starts a new round and has no uneliminated bordering agent, it declares existence of a hypertree.

Suppose A_j starts a new round with at least one uneliminated bordering agent. When the token backtracks to A_j, if A_j still has uneliminated bordering agents, it declares non-existence of hypertrees.

By Theorem 1, hypertree existence can be analyzed according to mutually exclusive and exhaustive types of boundary sets. We illustrate the above idea for Type 2 below. After specifying our algorithm in Section 8.1, we illustrate for Type 1 and Type 3.

Example 10 *Fig. 11 (a) shows the communication graph of the Type 2 boundary set from Example 6. Suppose A_0 starts the first round with tok^1. Since it satisfies border equality relative to A_1, A_0 announces*

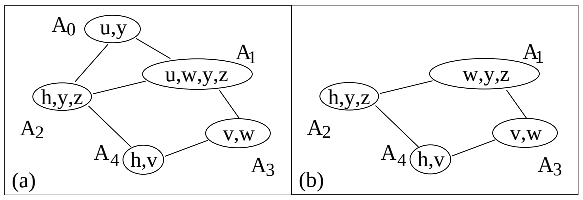

Figure 11: Hypertree existence recognition with a type 2 boundary set

elimination and passes a new token tok^2 to A_1. A_1 reduces its boundary as in (b) and starts the second round of DFT using tok^2. Since A_1 does not satisfy border equality relative to any of two bordering agents, it passes tok^2 to one of them, say, A_2. A_2 does not satisfy border equality relative to any of two bordering agents and passes tok^2 to A_4, that in turn passes to A_3.

A_3 cannot self-eliminate, nor does it have any unvisited agent to pass token to. Therefore, it returns tok^2 to A_4. A_4 returns tok^2 to A_2, that in turn returns to A_1. A_1 gets tok^2 back, while having two uneliminated bordering agents. It declares non-existence of hypertrees. Soundness of the claim can be seen from both Theorem 2 (through non-eliminability) and Theorem 1 (through Type 2 boundary set).

8 *HTBS* Algorithm Suite

We now specify a distributed algorithm suite *HTBS* for recognition of hypertree existence.

8.1 The Algorithm Suite

Activity at an agent is driven by the following messages.

- A notification *Eliminated* is sent by an agent that has been self-eliminated.
- A request $StartNewDFT(tok)$ calls the receiver to start a new round of DFT with token tok.
- A request $DFT(tok)$ calls the receiver to continue the current round of DFT with token tok.
- A *Report* is sent by an agent in response to $DFT(tok)$, signifying that either it has completed DFT or it has been visited in the current round. In either case, the current round of DFT backtracks to the caller.

We refer to an agent that runs a procedure by A_i and the sender of a message to A_i by A_c. Before responding to messages, every agent is initialized. A flag $state \in \{IN, OUT\}$ (set to IN) indicates whether A_i has been eliminated. A flag $nbsta(A_k) \in \{IN, OUT\}$ (set to IN) indicates the state of a bordering agent A_k. A variable *curtok* (set to *nil*) keeps a token value after it has visited A_i, and $visited(A_k)$ (set to $false$) indicates whether the token has visited a bordering agent A_k. A_i maintains its active boundary W_i as Y_i.

At the start of a new round, a remaining agent receives $StartNewDFT$. In the first round, a *leader* agent is arbitrarily selected and messaged externally. Its choice does not affect correctness of the outcome. When messaged, A_i does the following.

Procedure 7 (Response to StartNewDFT(tok))

```
1 if $A_c$ is a bordering agent,
2     $nbsta(A_c) = OUT$;
3     if there exists no $A_j$ with $nbsta(A_j) = IN$,
4         declare “a hypertree exists” and start halting;
5     else $Y_i = \emptyset$; for each bordering $A_k$ where $nbsta(A_k) = IN$, $Y_i = Y_i \cup I_{ik}$;
6 $curtok = tok$; $parent = nil$;
7 run DoDFT;
```

Proc. $DoDFT$ is run from either Proc. 7 (line 7) or Proc. 9 (line 4). In both cases, A_i has at least one remaining bordering agent. When from Proc. 7 where A_c is external, all bordering agents of A_i are IN. If A_c is a bordering agent, A_i must have remaining bordering agents since otherwise A_i would have halted in line 4. When from Proc. 9, A_c is a remaining bordering agent. Note when executing from Proc. 7, A_i has $parent = nil$, while when executing from Proc. 9, $parent = A_c$.

During $DoDFT$, A_i looks for A_j with border equality. If found, A_i sends $Eliminated$ to each remaining bordering agent and $StartNewDFT$ to A_j. Otherwise, A_i looks for a remaining bordering agent A_k that differs from *Ac*. If found, A_i sends $DFT(tok)$ to A_k. If no such A_k exists, A_i sends *Report* to A_c.

Procedure 8 (DoDFT)

```
1 if there exists $A_j$ with $nbsta(A_j) = IN$ and $Y_i = I_{ij}$, // self-eliminate
2     $state = OUT$;
3     for each bordering $A_k \neq A_j$ where $nbsta(A_k) = IN$, send $Eliminated$ to $A_k$;
4     send $StartNewDFT(curtok + 1)$ to $A_j$;
5 else parent = $A_c$; // no IN agent satisfies $Y_i = I_{ij}$
6     for each bordering $A_k \neq$ parent where $nbsta(A_k) = IN$, set $visited(A_k) = false$;
7     if there exists $A_k \neq$ parent where $nbsta(A_k) = IN$ and $visited(A_k) = false$,
8         send message $DFT(curtok)$ to $A_k$;
9     else send Report to parent;
```

When A_i receives $Eliminated$, it sets $nbsta(A_c) = OUT$. When A_i receives $DFT(tok)$ from A_c, it performs the following. Line 1 is run when A_i has been visited in the current round and the *Report* corresponds to backtracking. Otherwise, A_i continues with the current round.

Procedure 9 (Response to DFT(tok))

```
1 if $curtok = tok$, send Report to $A_c$; // tok is not fresh
2 else $curtok = tok$; // tok is fresh to $A_i$
3     $parent = A_c$; $visited(A_c) = true$;
4     run DoDFT;
```

After A_i sends $DFT(tok)$ to A_j, it may receive a *Report* in reply. In response, A_i looks for another remaining bordering agent A_k other than A_c to continue the current round of DFT. If A_k is not found and A_i is the sender of $StartNewDFT$ for the current round, it declares no hypertrees. Otherwise, it backtracks to A_c with a *Report*.

Procedure 10 (Response to Report)

```
1 $visited(A_j) = true$;
2 if there exists $A_k \neq$ parent such that $nbsta(A_k) = IN$ and $visited(A_k) = false$,
3     send message $DFT(curtok)$ to $A_k$;
4 else // no unvisited bordering agent
5     if $parent = nil$, declare “no hypertrees” and start halting;
6     else send Report to parent;
```

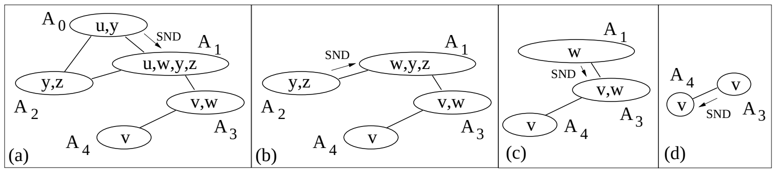

Figure 12: Execution of *HTBS* with a Type 1 boundary set

Example 11 *Fig. 12 (a) shows the communication graph of the Type 1 boundary set from Example 5. Suppose A_0 is the leader receiving StartNewDFT(tok^1). Since it satisfies border equality relative to A_1, A_0 sends Eliminated to A_1 and A_2, and sends StartNewDFT(tok^2) (shown as SND) to A_1. In response, A_1 updates its active boundary as in (b). Since A_1 does not satisfy border equality relative to A_2 and A_3, it sends DFT(tok^2) to one of them, say, A_2. Border equality holds for A_2. Hence, it sends Eliminated to A_1, followed by StartNewDFT(tok^3).*

In response, A_1 updates its active boundary again as in (c). Now, it satisfies border equality relative to A_3. It sends Eliminated and StartNewDFT(tok^4) to A_3. In response, A_3 updates its active boundary as in (d). It sends Eliminated and StartNewDFT(tok^5) to A_4. A_4 has no uneliminated bordering agent and declares hypertree existence. Soundness of the claim can be seen from Theorem 2 (eliminability) or Theorem 1 (Type 1).

During DFT, the active boundary of an agent may be reduced but the border between any pair of agents never changes. Furthermore, sending both *Eliminated* and *StartNewDFT* to a same agent is unnecessary. For simplicity of presentation, such optimization is omitted.

Example 12 *Fig. 13 (a) shows the communication graph of the Type 3 boundary set from Example 7. Suppose A_0 is the leader receiving StartNewDFT(tok^1). It sends Eliminated to A_1 and A_2 and sends*

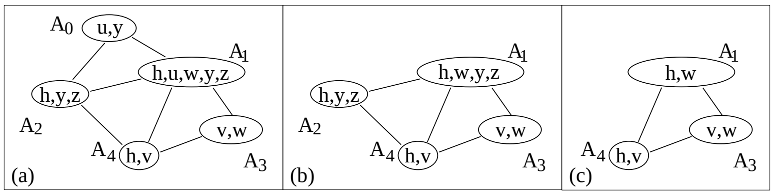

Figure 13: Execution of *HTBS* with a Type 3 boundary set

StartNewDFT(tok^2) to A_1. A_1 updates its boundary as in (b). Since A_1 does not satisfy border equality relative to any bordering agent, it sends DFT(tok^2) to one of them, say, A_2. A_2 sends Eliminated to A_1 and A_4 and StartNewDFT(tok^3) to A_1.

After updating its boundary as in (c), A_1 sends DFT(tok^3) to, say, A_3. In response, A_3 sends DFT(tok^3) to A_4 that in turn sends to A_1. Since token tok^3 is not fresh to A_1, it sends Report to A_4. In response, A_4 sends Report to A_3 that in turn sends to A_1. A_1 has no unvisited bordering agent and declares no hypertrees. Soundness of the claim can be seen from Theorem 2 (non-eliminability) or Theorem 1 (Type 3).

In Procs. 7 and 10, how to perform halting is not elaborated. One method is for the declaring agent to send each bordering agent the claim, and halt. Each receiving agent relays to each bordering agent except its sender and halts.

8.2 Soundness, By-product, Complexity, and Agent Privacy

HTBS concludes with the claim of either "a hypertree exists" or "no hypertrees". Its soundness and completeness follow directly from Theorem 2 as stated below.

Corollary 1 *A MAS with the boundary set W has a hypertree iff $HTBS$ terminates with "a hypertree exists".*

Retrospectively, *DPMST* was developed first. As it assumes hypertree existence, $HTBS$ was developed subsequently with the main focus on recognizing existence. After $HTBS$ was formulated as above, we discovered to our surprise that it has a significant by-product as illustrated below.

Example 13 *In Example 11, a total of four $StartNewDFT$ messages are sent between agents (an additional one comes to the leader externally). These messages are labelled as SND in Fig. 12. Pathways of these messages define a boundary based JT (see Fig. 8 (d)).*

When A_i sends $StartNewDFT$ to A_j ($j \neq i$), a sender-receiver relation forms. Example 13 suggests that, if $HTBS$ concludes with a positive claim, $StartNewDFT$ sender-receiver relations define a boundary based JT. The theorem below generalizes this observation.

Theorem 3 *If a MAS with the boundary set W has a hypertree, then agent adjacency in a hypertree is defined by $StartNewDFT$ sender-receiver relations during HTBS.*

Theorem 3 means that as long as agents keep track of $StartNewDFT$ sender-receiver relations, upon claim of "a hypertree exists", a hypertree is immediately specified without additional computation.

Next, we analyze complexity. Let e be the number of pairs of bordering agents. In each round of traversal, $O(e)$ messages are passed. HTBS concludes in $O(\eta)$ rounds. Hence, the computation time up to declaration is $O(e\ \eta)$. The halting process takes $O(r\ \eta)$ messages, where r is the maximum number of bordering agents per agent. Therefore, the overall time complexity is $O((e+r)\ \eta)$.

Finally, we consider agent privacy. $HTBS$ solves the reformulated Problem 2 (as well as the reformulated Problem 1). Hence, only loss on agent identity, bordering relation, and shared variable are possible. Since messages contain no information on shared variable, there is no loss on shared variables. Since messages are passed between bordering agents only and message argument is a token, there is no loss on agent identity and bordering relation. Note that the inference on agent adjacency during *DPMST* is not possible. The reliability of Rule to infer adjacency in Section 6.2 depends on the *hop* value. Since no similar path length information is relayed during $HTBS$, the inference is impossible.

9 Experimental Evaluation

The experimental study achieves the following purposes. First, it provides empirical evidence about soundness of *DPMST* and $HTBS$. Second, it compares their effectiveness in privacy preservation with alternative hypertree methods. Third, it compares computational costs with relevant methods. Fourth, it evaluates the chance of hypertree existence for an arbitrary environment decomposition. Fifth, it suggests promising direction on hypertree construction for environment decompositions without hypertrees.

9.1 Setup of Experiment on Soundness, Privacy Loss and Efficiency

A total of 405 environments are simulated using WebWeavr.[1] Each of them is decomposed over between 96 and 112 agents ($96 \leq \eta \leq 112$). This scale is sufficiently large so that a further scaling up differs quantitatively rather than qualitatively. Among the 405 environments, 137 are Type 1, 135 are Type 2, and 133 are Type 3. Hence, possible types are covered exhaustively. Environments also differ in densities of communication graphs. Each environment is at one of three possible levels of density, *sparse*, *denser*, and *densest*, which corresponds to about 5, 16, and 27 borders per agent on average. Each subenvironment has between 2 and 37 variables. Using the digital equipment diagnosis example [Xia02] as the reference, where ratios of private versus shared variables are between 3.6:10 and 8.2:10, we set the ratio for each subenvironment randomly between 3:10 and 10:10. Subenvironment sizes and ratios of private versus shared variables are so chosen such that further scaling up differs quantitatively rather than qualitatively. Table 1 summarizes the distribution of these environments and their indexing (to be referenced below).

[1]WebWeavr is a Java-based toolkit for graphical models and is available at http://www.socs.uoguelph.ca/˜yxiang/.

Table 1: Experimental environments

Env Type	Type 1			Type 2			Type 3		
Density	Sparse	Denser	Densest	Sparse	Denser	Densest	Sparse	Denser	Densest
No. Env	45	46	46	45	45	45	45	44	44
Index	1-45	46-91	92-137	138-182	183-227	228-272	273-317	318-361	362-405

To evaluate the relative effectiveness of DPMST and HTBS, we selected alternative algorithms as follows. Since DPMST and HTBS are intended for privacy preserving hypertree construction, the relevant algorithms reviewed in Section 3.2 are the immediate candidates, including Action-GDL, DCTE, and COORD-Plus. COORD-Plus uses one private variable per agent (Section 4.1). This can be improved as justified by Propositions 1 and 5. We therefore replace COORD-Plus with a method COORD that modifies COORD-Plus by not using the private variables.

Since DPMST is based on distributed MST construction, we considered alternative algorithms surveyed in Section 5.1. They can be divided into three types, the GHS type including GHS and its improvements [Awe87, FM95, GKP98], the approximate type [KP08], and the parallel type [NCKB12]. As discussed in Section 5.1, approximate MST algorithms are not applicable to our problem and parallel algorithms have much worse privacy loss than the GHS type. Hence, we eliminated these two types from candidates for comparison. Within the GHS type, although more recent algorithms improve efficiency relative to GHS, they suffer the same privacy loss as GHS due to the same technique used to handle nondistinct link weights. As our primary focus is on privacy preservation rather than efficiency of distributed MST construction, we have chosen to use GHS as the representative for this type of algorithms.

In addition, a centralized algorithm, termed CENTRA, is used as the baseline for privacy loss. CENTRA collects the subenvironment of every agent to a single agent A_i. A_i determines hypertree existence and if so builds a hypertree (see Section 9.3 [Xia02]). It then informs each agent of its hypertree neighbors. As shown in an earlier work [MPB$^+$06] and also verified by our experimental results below, decentralized approaches do not automatically outperform centralized approaches with respect to privacy loss. Hence, the use of a centralized algorithm as the baseline is justified.

In summary, seven alternative algorithms are compared, Action-GDL, CENTRA, COORD, DCTE, DPMST, GHS, and HTBS. Note that since CENTRA collects the subenvironment from every agent to a single agent A_i, the maximum privacy leak occurs to A_i but none to other agents. To make the result independent of the choice of A_i, we measure the system privacy loss using the expected value. The expected normalized system privacy loss for CENTRA is $E(NSPL) = 1/\eta$. Extending the analysis in Section 3.2, COORD has the same $E(NSPL)$ as CENTRA on agent identity and adjacency and on identity of shared variable, but incurs no loss on domain of shared variable and on identity and domain of private variable.

Action-GDL, DCTE, GHS, DPMST, and HTBS are implemented in Java using WebWeavr, running in simulated MAS environments. Each message transmission takes 1 time unit. The computational cost (runtime) is measured by the number of time units. For all methods, an arbitrary root agent is externally specified, that starts the clock. Each other agent is initially inactive until a message is passed to it according to the algorithm. For Action-GDL, this root starts depth-first search and becomes the root of the pseudotree. For DCTE, this agent generally does not end up as the root of the spanning tree. For GHS, this root is the first awakened fragment. For DPMST, this root is the MST root and for HTBS it is the leader.

During execution of each algorithm, each agent keeps track what are learned from messages on unknown agent identity and adjacency, and on unknown variable identity and domain. Each DPMST agent also uses the rule in Section 6.2 to infer agent adjacency. After the algorithm halts, information is collected from each agent to compile the normalized loss NSPL.

GHS and DPMAT are intended for hypertree construction assuming existence and were run on 137 Type 1 environments. Action-GDL, DCTE, and HTBS were run on all 405 environments. $E(NSPL)$ from CENTRA and COORD are derived and used for comparison with other methods.

9.2 Results on Soundness and Privacy Loss

HTBS is the only distributed algorithm run for existence recognition. For all 268 environments of Type 2 and Type 3, HTBS correctly recognized non-existence of hypertrees. For the remaining 137 Type 1 environments, it constructed hypertrees. GHS and DPMST were run on 137 Type 1 environments and constructed hypertrees in all. Action-GDL and DCTE constructed hypertrees in all 405 environments. Hence, all alternative methods completed intended tasks correctly. Note that HTBS is strictly bounded by the environment decomposition while Action-GDL and DCTE are not (variables can be added to the subenvironment of an agent). Therefore, Action-GDL and DCTE constructed hypertrees for Type 2 and Type 3 environments when HTBS declares non-existence.

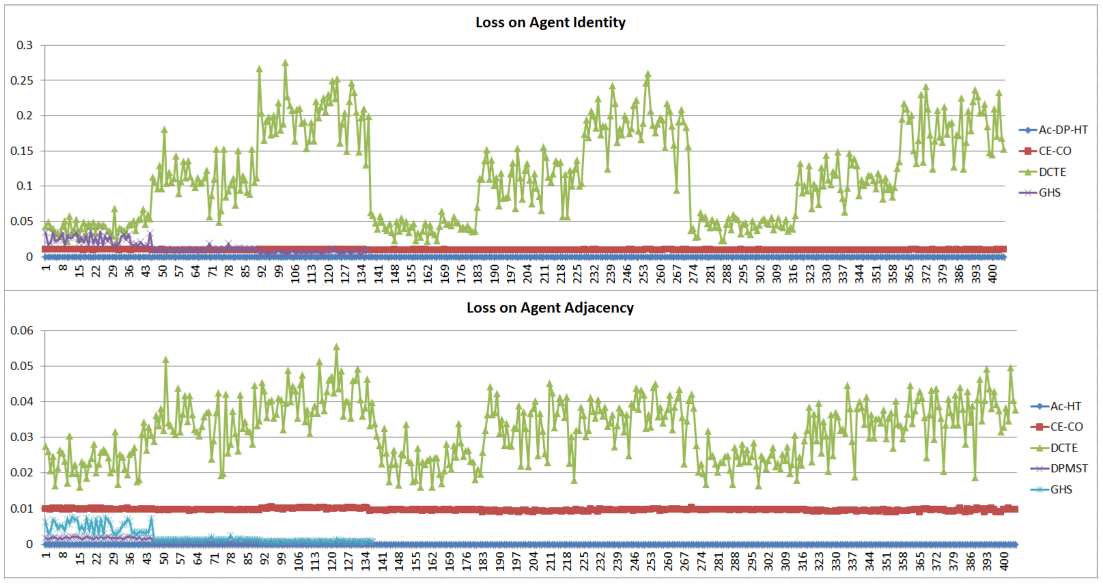

Figure 14: Normalized system privacy losses on agent identity and adjacency

Fig. 14 plots NSPL on agent identity (top) and adjacency (bottom), where the horizontal axis is labeled by environment index. When different methods yield the same NSPL, their plots are combined. For instance, all of Action-GDL, DPMST and HTBS incur no loss on agent identity. A single curve is shown and labeled as Ac-DP-HT. Abbreviations (first two letters) in figure legends are listed below.

$$Ac : Action-GDL;\ CE : CENTRA;\ CO : COORD :\ DC : DCTE;\ DP : DPMST;\ GH : GHS;\ HT : HTBS.$$

GHS and DPMST were run for Type 1 environments and their curves extend up to index 137.

DCTE has the highest loss on both agent identity and adjacency, and the loss is consistently higher than the baseline (CENTRA). The loss grows as the density level increases, but is insensitive to the type of environment. GHS has the loss on agent identity (top) close to the baseline, but its loss on agent adjacency (bottom) is consistently lower than the baseline. In both cases, the loss decreases as the density level increases. Action-GDL, DPMST and HTBS incur no loss on agent identity, and Action-GDL and HTBS incur no loss on agent adjacency.

Fig. 15 plots NSPL on private variable identity (top) and domain (bottom). Since losses on private variable identity by Action-GDL, CENTRA, and DCTE are significantly different, their losses are plotted in log10 (top). Losses by COORD, DPMST, GHS, and HTBS are zero and are not shown, since $\lim_{loss\to 0}\ log_{10}(loss) = -\infty$.

DCTE (top) has the maximum possible loss ($NSPL = 1$) on private variable identity. This is due to global variable propagation during JT construction. The original method [PGM05] did not explicitly differentiate identity from domain in variable propagation. Our implementation of DCTE propagates variable IDs first and only propagate domains as needed. Since a private variable is contained in a single agent, it is never added to another agent for running intersection. Our implementation therefore did not

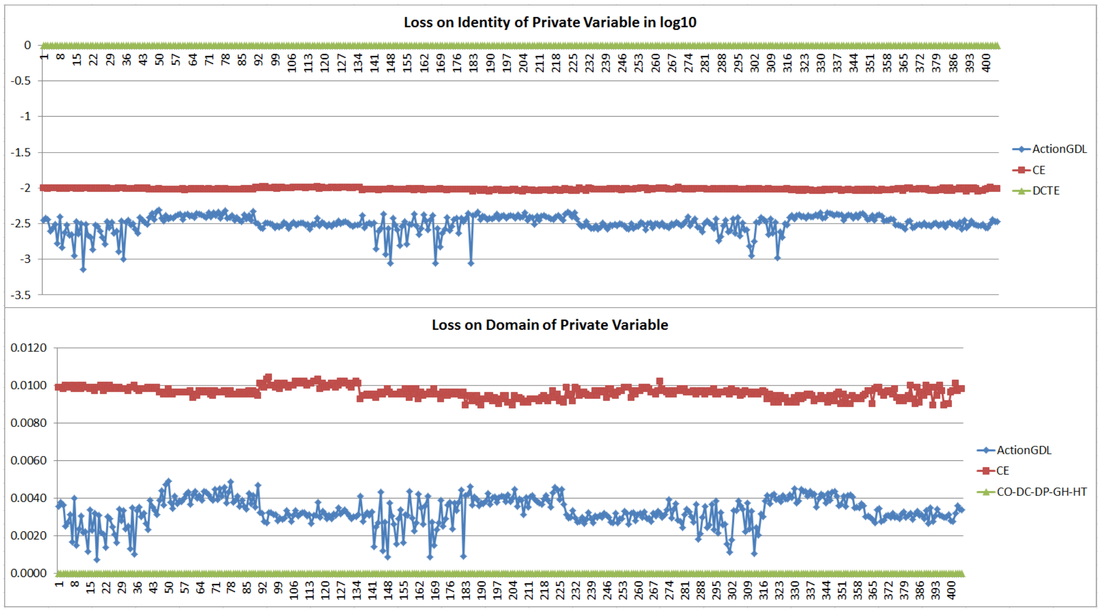

Figure 15: Normalized system privacy losses on identity and domain of private variable

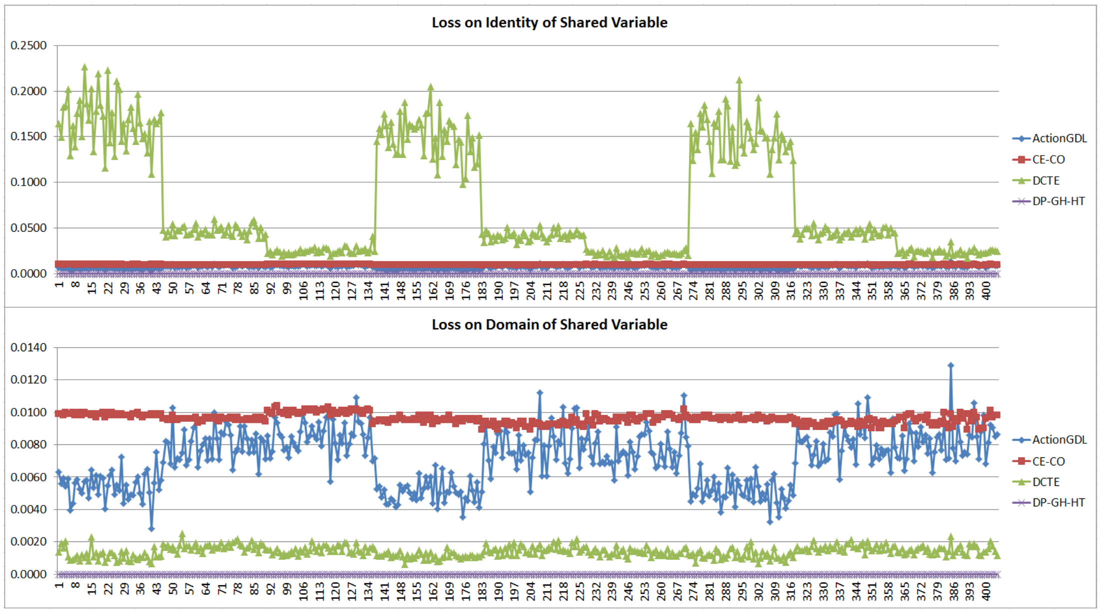

Figure 16: Normalized system privacy losses on identity and domain of shared variable

propagate its domain, resulting in zero loss. This experimental result suggests that their IDs need not be propagated. Hence, DCTE can be improved to reduce loss on identity of private variable to none.

Losses by Action-GDL on both private variable identity and domain are consistently lower than the baseline. COORD, GHS, DPMST, and HTBS incur no loss in either case.

Fig. 16 plots NSPL on the identity (top) and domain (bottom) of shared variable. Losses by Action-GDL are identical on both and difference in the outlook is due to different scales at y-axis. Its losses are generally lower than the baseline. DCTE's loss on shared variable identity is consistently much higher than CENTRA, but its loss on shared variable domain is much smaller than CENTRA. The striking difference is due to global variable ID propagation, but domain propagation occurs at a much smaller scale. DCTE's loss on variable identity decreases with the density level, but its loss on variable domain does not vary with the density. On the other hand, the loss by Action-GDL grows with the density level. No loss is incurred by GHS, DPMST and HTBS on both variable identity and domain.

Table 2 summarizes privacy loss by alternative methods. Losses by CENTRA and COORD are expected values. Loss on agent adjacency by DPMST is produced soly due to the rule of inference.

Table 2: Summary of privacy loss by alternative algorithms

	Agent		Private Var		Shared Var	
	Id	Adj	Id	Dom	Id	Dom
CENTRA	$1/\eta$	$1/\eta$	$1/\eta$	$1/\eta$	$1/\eta$	$1/\eta$
COORD	$1/\eta$	$1/\eta$	0	0	$1/\eta$	$1/\eta$
Action-GDL	0	0	x	x	x	x
DCTE	x	x	x	0	x	x
GHS	x	x	0	0	0	0
DPMST	0	x	0	0	0	0
HTBS	0	0	0	0	0	0

The significance of comparison between GHS and DPMST on agent adjacency loss is evaluated by a Friedman test. Rank sums are 137 and 274 respectively and the critical value for $\alpha = 0.01$ is 27.23. Hence, the null hypothesis is rejected at the $\alpha = 0.01$ level of significance. Neither method incurs loss on private and shared variable. Since GHS incurs loss on agent identity but DPMST does not, and loss by DPMST on agent adjacency is significantly less, DPMST has superior privacy than GHS.

Among the alternatives, HTBS is the only one that incurs no privacy loss. DPMST has the lowest privacy loss among all alternatives, except HTBS.

9.3 Results on Runtime

Runtimes by Action-GDL, DCTE, GHS, DPMST, and HTBS are plotted in Fig. 17. DCTE has the highest cost due mainly to spanning tree construction. Let d be the diameter of the dependency graph and r be the maximum node degree. Starting at an arbitrary node, it takes DCTE $O(d\ g)$ time to activate the spanning tree root, and another $O(d\ g)$ time for each node to agree on the root, which completes the spanning tree construction. Before $2dg$ time, it's impossible to ensure that the spanning tree has been distributively constructed. Hence, the cost of DCTE is lower-bounded by $2dg$ time.

Action-GDL has about the same runtime as DPMST for the denser level, does better for the sparse, and does worse for the densest. Although GHS is the fastest due to parallel processing, HTBS's cost is only slightly higher than GHS but is consistently lower than all other alternatives.

The relative cost of DPMST and HTBS is somewhat surprising. From the complexity $O((2d+r)\eta)$ for DPMST and $O((e+r)\eta)$ for HTBS, DPMST is expected to be more efficient. To the contrary, it takes on average three times longer than HTBS in Type 1 environments. This is attributed to the following. For *HTBS*, after an agent is self-eliminated, it no longer participates in further computation, until propagation of the final claim. Hence, the active boundary set continues to shrink during the process. For *DPMST*, after a node is added to the MST, it continues to participate and may relay many

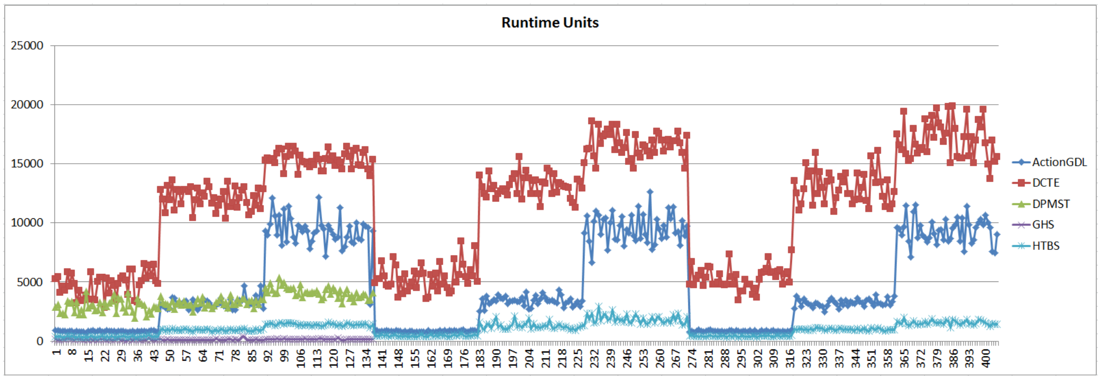

Figure 17: Runtime measured in time units

Expand/*Report* messages, until no outgoing link is reachable from it. This contributes to a longer runtime.

9.4 Likelihood of Hypertree Existence

This experiment evaluates the likelihood of hypertree existence with an arbitrary boundary set. It is equivalent to evaluate the likelihood of a JT for an arbitrary cluster graph. To control the scale, we set the generating set for clusters at size 10 and let each cluster be a proper subset. A cluster has at least 2 elements to allow intersections with different clusters over different elements. All clusters in a cluster graph have the same size, which controls the average degree of interaction between clusters. For each cluster graph of cluster size k, clusters are randomly selected from $C(10,k)$ distinct candidates. For $k=2$ the number of candidates is 45, and for $k=5$ the number is 252. For each cluster graph, we select 10 clusters. For each cluster size $k=2,3,4,5,6,7$, we generate 1000 cluster graphs.

Whether a cluster graph has a JT can be determined by converting it uniquely into an undirected graph and test its chordality (see [Xia02] for details). This is due to the well-known fact that the cluster graph has a JT if and only if the undirected graph is chordal. Each of the 6000 cluster graphs were tested for JT existence. The percentage of cluster graphs with JTs for each group (1000 graphs of the same cluster size) is shown in Fig. 18.

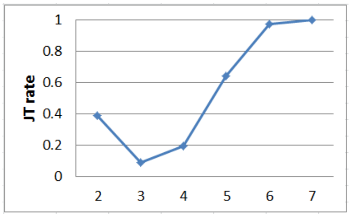

Figure 18: Percentage of cluster graphs with JTs. The x-axis is labelled by cluster size k.

When $k=2$, 39% of the cluster graphs have JTs, but the rate drops to 9% when $k=3$. This is because many undirected graphs with $k=2$ have no loops and hence are chordal. As k increases to 3, more loops without chords formed and resultant graphs are nonchordal. As k continues to increase, more chords are added to loops, rendering more chordal graphs. When k increases to 7, 99% of the cluster graphs have JTs.

This result demonstrates that if the communication graph is dense (such as the case $k=7$), it is highly likely that the boundary set has a hypertree. Otherwise, the boundary set is less likely to have a hypertree. In that case, no matter how boundaries are arranged into a tree organization, it does not ensure correct inference. Hypertree recognition is necessary to detect such situations.

9.5 When Boundary Set Has No Hypertrees

Next, we consider the third issue raised in Section 1 on how to modify an environment decomposition that has no hypertrees. From Theorem 1, a boundary set of Type 2 (Fig. 9) or Type 3 (Fig. 10) has no hypertrees. We demonstrate that both can be modified to have a hypertree by incurring a limited privacy loss on shared variable.

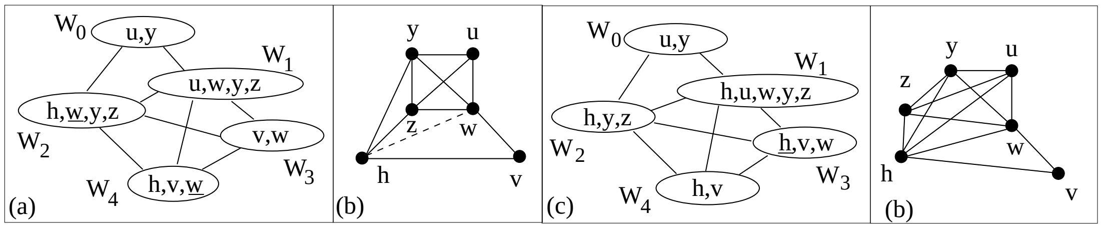

Figure 19: (a) A boundary set. (b) The boundary graph of (a). (c) Another boundary set. (d) The boundary graph of (c).

Fig. 19 (a) shows a boundary set modified from Fig. 9. Variable w originally shared by A_1 and A_3 is now also shared by A_2 and A_4 (shown by underline). The new boundary graph is in (b) where a (dashed) link is added as the result. The graph is chordal with three cliques and each is boundary contained. Hence, the modified boundary set has a hypertree.

Fig. 19 (c) shows a boundary set modified from Fig. 10. Variable h originally shared by A_1, A_2 and A_4 is now also by A_3. The boundary graph (d) is identical to Fig. 10 and is chordal. What differs is that it is now boundary contained. Hence, the modified boundary set has a hypertree.

Our earlier results show that for the 268 Type 2 and Type 3 environments (indexed 138 to 405), HTBS incurred no privacy loss while Action-GDL and DCTE incurred loss up to and higher than the baseline. One may attribute the superior privacy of HTBS to the lesser task to recognize hypertree non-existence and infer that the loss by Action-GDL and DCTE is necessary as they constructed hypertrees. Although our demonstration above indicates that some privacy loss is unavoidable during construction, we claim that up to the baseline (as Action-GDL) and higher than the baseline (as DCTE) loss is not necessary. Our experimental results clearly support the claim. First, significant difference between losses by Action-GDL and DCTE means that the higher than baseline loss can be avoided. Second, for the 137 Type 1 environments, HTBS performed the same task as Action-GDL and DCTE and their relative performance on privacy loss is about the same as for Type 2 and Type 3. This suggests the following future research. Extend HTBS to environment decompositions that have no hypertrees by modifying the decompositions and constructing hypertrees. The direction is promising in resolving the third issue whiling incurring much less privacy loss than Action-GDL and DCTE.

10 Conclusion

A competitive agent organization needs to support both sound inference and agent privacy. Hypertrees have the former property and the potential to the latter. However, existing hypertree frameworks do not deliver the potential for privacy and leak three types of private information that can be protected.

This paper presents six main contributions to address the limitation. First, we identified three types of agent privacy that are compromised by existing hypertree techniques and defined a set of measurements to quantify each type of privacy loss. Second, we proposed the boundary set based and MST based problem reformulations that allow development of new hypertree algorithms that fundamentally protect agent privacy on private and shared variables. Third, we proved a mutually exclusive and exhaustive typing of boundary sets: a foundation for algorithmic study of hypertree existence. Fourth, we developed DPMST for hypertree construction, taking advantage of the MST based problem reformulation. Although slower than GHS, DPMST has no loss on agent identity, incurs significantly lower loss on agent adjacency, and remains efficient. It has the lowest privacy loss among alternatives, except HTBS. Fifth, we developed HTBS for existence recognition and hypertree construction. It has no privacy loss, it is efficient, and its cost is lower than alternative methods except GHS. Sixth, we conducted extensive experiments demonstrating the superiority of DPMST and HTBS in agent privacy.

As far as tasks of existence recognition and hypertree construction are concerned, HTBS dominates DPMST. This is because (1) DPMST handles only one task while HTBS handles both, (2) HTBS has no privacy loss while DPMST has non-zero loss on agent adjacency, and (3) HTBS runs faster. Still, DPMST holds its value both algorithmically and practically. It is the first distributed MST algorithm that protects node identities and has significantly less loss on node adjacency than GHS (and its improvements). It provides a methodology-wise significantly different alternative and opens the door for further improvement of the methodology. Since DPMST assumes hypertree existence, actual existence generally has to be established by other means. However, there are cases where the hypertree existence is known and hence DPMST is applicable. For instance, when an existing MAS with a hypertree is expanded such that each new agent borders exactly one existing agent, a hypertree exists for the expanded environment.

Performance in constraint reasoning can be improved by selecting the pseudotree used [MTB$^+$05]. DPMST and HTBS do not try to control topology of the hypertree produced. One reason is that the number of alternative pseudotrees for a given dependency graph is large. Every node can be the root to produce at least one distinct pseudotree. On the other hand, the number of alternative hypertrees for a given environment decomposition is often small, unless many agent borders are identical. For instance, the hypertree for Fig. 1 (a) is unique. Secondly, unlike a pseudotree where the root is fixed and inference time is thus dependent on the pseudotree, a hypertree does not have a rigid root. That is, inference on a hypertree can be started at any agent. Although the actual root influences the inference time, the root choice needs not be made at the time of hypertree construction.

Privacy preserving hypertree construction involves three essential issues: existence recognition, construction, and modification of environment decompositions that have no hypertrees. HTBS addresses the first. Both DPMST and HTBS address the second. We have demonstrated that an environment decomposition without hypertrees can be modified to have one. A promising direction for a general solution of the third issue is suggested, which extends HTBS to reduce privacy loss potentially well below existing methods such as Action-GDL and DCTE. DPMST and HTBS involve all agents in a MAS. For MASs that dynamically expand and shrink, incremental recognition and construction may be obtained through extensions.

In distributed constraint optimization, a metric that combines the value of a solution with the loss of privacy has been proposed [SDMY08]. It allows an agent to quit problem solving when anticipated privacy loss is higher than the value of a solution. This idea is useful when an agent must trade solution value with privacy. Such tradeoff is not a concern when a solution (recognition of hypertree existence or its construction when existing) can be found efficiently without any loss of privacy (as by HTBS) or with very limited loss (as by DPMST). It may become relevant as we tackle the third issue outlined above in future research.

Both DPMST and HTBS assume an externally specified root or leader agent. The soundness of both algorithms are independent of the choice of this agent. It is sometimes desirable to elect the root/leader agent while preserving the three types of privacy. We leave the feasibility analysis of such an election and, if positive, its algorithmic development for future research.

Appendix: Proofs

Proof of Proposition 1

We show that running intersection holds in T'. Let C' and Q' be non-adjacent clusters in T' such that $C' \cap Q' \neq \emptyset$, and X' be a cluster on the path between C' and Q'. Let C, Q, X be clusters in T corresponding to C', Q', X', respectively. We show that $C' \cap Q'$ is contained in X', namely, $C' \cap Q' \subseteq X'$.

Since T is a JT, $C \cap Q$ is contained in X, i.e., $C \cap Q \subseteq X$. Because C and Q are boundaries (made of shared variables), it follows $C' \cap Q' = C \cap Q$. That is, $C' \cap Q' \subseteq X$. Since X is a boundary, we have $X' = X \cup Y'$, where Y' is the set of private variables in subenvironment X'. Hence, $X \subseteq X'$. From $C' \cap Q' \subseteq X$ and $X \subseteq X'$, it follows that $C' \cap Q' \subseteq X'$. $\square$

Proof of Proposition 3

By assumption, the transmission of each message takes at most one time unit. Combining Eqns. (7) and (9) and substituting t_4 by t, the result follows. $\square$

Proof of Proposition 5

Suppose that no JT exists with boundaries in W as clusters, but a JT T' exists with subenvironments in Ω as clusters. Then for every pair of nonadjacent clusters in T' such that $C' \cap Q' \neq \emptyset$, and a cluster X' on the path between C' and Q', it holds that $C' \cap Q' \subseteq X'$.

Let T be a cluster tree with boundaries in W as clusters such that it is isomorphic to T' with each boundary mapped to the corresponding subenvironment. Let C, Q, X be clusters in T corresponding to C', Q', X', respectively. Since C and Q are boundaries, $C \cap Q = C' \cap Q' \subseteq X' = X \cup Y'$, where Y' is the set of private variables in subenvironment X'. From $C \cap Q \subseteq X \cup Y'$ and $C \cap Q \cap Y' = \emptyset$, we obtain $C \cap Q \subseteq X$. That is, running intersection holds in T and T is a JT: a contradiction to the assumption. $\square$

Proof of Lemma 1

From subcondition 1, a JT T exists and no two clusters in T are comparable. We prove the claim by contradiction. Suppose there exists a cluster Q in T such that for every boundary $W_i \in W$, $Q \neq W_i$.

From subcondition 2, there exists a boundary W_i such that $Q \subseteq W_i$. Since $Q \neq W_i$, it follows that $Q \subset W_i$. Because BG is the boundary graph, W_i is complete in BG. Therefore, there exists a clique C_i in BG such that $W_i \subseteq C_i$. Since clusters of T are cliques in BG, C_i must be a cluster in T. From $Q \subset W_i$ and $W_i \subseteq C_i$, we have $Q \subset C_i$. That is, T contains two comparable clusters: a contradiction. Hence, every cluster in T is a boundary. $\square$

Proof of Theorem 1

[Necessary Condition] Suppose a JT H exists whose clusters are subenvironments. Extending Proposition 1, if private variables are removed from each cluster, the resultant cluster tree T is still a JT, whose set of clusters is W. Since T is a JT, the boundary graph BG is chordal and subcondition 1 holds. Since each clique of BG is a cluster in T and each cluster of T is a boundary, subcondition 2 follows.

[Sufficient Condition] Suppose both subconditions hold. We prove by construction. Since BG is chordal, a JT T exists whose clusters are cliques of BG. By Lemma 1, every cluster in T is a boundary. Hence, for every cluster C in T such that $C = W_i$ for some $W_i \in W$, we associate C with agent A_i.

It is possible that not every agent has been associated with a cluster in T yet. In that case, consider such an agent A_i whose boundary is W_i. Since W_i is complete in BG, there exists a cluster C in T such that $W_i \subseteq C$. Add a new cluster W_i to T, making it adjacent to cluster C only, and associate the new cluster with A_i. Repeat this for each remaining agent, until each agent is associated with a cluster in T.

Next, for each agent, add its private variables to its associated cluster in T. The resultant T is a JT agent organization with each cluster being a subenvironment. $\square$

Proof of Theorem 2

Based on Proposition 1, it suffices to prove the theorem relative to a boundary based JT.

[Necessary Condition] Suppose a boundary based JT T exists whose clusters are boundaries in W. We show that boundaries in W can be eliminated iteratively using T as a reference structure.

Since T is a tree, there exists a leaf cluster W_i. Let W_j be the adjacent cluster of W_i. Since T is a JT, for every W_k such that $W_k \neq W_i$ and $W_k \neq W_j$, we have $W_i \cap W_k \subseteq W_j$. Hence, $W_i \subseteq W_j$ and W_i can be eliminated from W relative to W_j to obtain W'_j and the reduced boundary set W'. Cluster W_i can also be removed from T, with cluster W_j replaced by W'_j and resultant cluster tree denoted by T'.

Now the set of clusters of T' is W'. Since W'_j differs from W_j only in terms of variables that W_j shares uniquely with W_i, T' is still a JT. Hence, the above operation can be applied repeatedly, until two boundaries are left. Since the reduced W' is a well-defined boundary set, the two boundaries must be identical and the last elimination results in $\{\emptyset\}$.

[Sufficient Condition] Suppose W can be eliminated iteratively into a singleton. Denote the sequence of reduced boundary sets as

$$W^{\eta}, W^{\eta-1}, \ldots, W^2, W^1,$$

where η is the number of agents, $W^{\eta} = W$, $W^1 = \{\emptyset\}$, and superscript counts the number of boundaries.

Each W^i $(i > 1)$ is a well-defined boundary set. The elimination process can be viewed as follows. To produce W^x from W^{x+1}, eliminate a $W_i \in W^{x+1}$ relative to a $W_j \in W^{x+1}$. Remove from W_j variables that it shares with W_i but not with any other $W_k \in W^{x+1}$. The result is W^x. We show below that boundaries in each W^x, for $x = 2, ..., \eta$, can be organized into a JT.

It is trivially true for $x = 2$, as can be seen from Example 9. We assume that it is true for $x = n$, and consider the case $x = n+1$. Suppose when W^n is derived from W^{n+1}, $W_i \in W^{n+1}$ is eliminated relative to $W_j \in W^{n+1}$ and W_j is reduced to $W'_j \in W^n$. By inductive assumption, boundaries in W^n can be organized into a JT T^n. If the cluster W'_j in T^n is replaced by W_j, the resultant cluster tree is still a JT, since $W_j \setminus W'_j$ is not shared by any other cluster in T^n. Next, add cluster W_i to T^n, and make it adjacent to W_j. Since $W_i \subseteq W_j$, the resultant cluster tree is still a JT, and its clusters are exactly the boundaries in W^{n+1}. Hence, boundaries in W^{n+1} can be organized into a JT.

From the above induction, boundaries in $W = W^{\eta}$ can be organized into a JT. □

Proof of Theorem 3

It suffices to show that the sender-receiver relations define a boundary based JT for W. Assume that W has a boundary based JT T. Since T is a cluster tree, there exists a leaf cluster $W_i \in W$ in T, that is adjacent to a single cluster $W_j \neq W_i$ in T. Because T satisfies running intersection, $W_i \cap W_k \subseteq W_j$ holds for every $W_k \in W$, where $k \neq i$. It can be equivalently written as $W_i \cap W_k \subseteq W_i \cap W_j = I_{ij}$ for every $W_k \neq W_i$. It follows that

$$\bigcup_{k \neq i} (W_i \cap W_k) \subseteq I_{ij}.$$

Since W_i is the boundary of A_i, $\bigcup_{k \neq i}(W_i \cap W_k) = W_i$ and the above becomes $W_i \subseteq I_{ij}$. As I_{ij} is the border between A_i and A_j, we have $W_i \supseteq I_{ij}$ which yields $W_i = I_{ij}$.

Let Q be the set of leaf clusters in T and S be the set of links between clusters in T. Let T' be another boundary based JT of W with the corresponding Q' and S'. In general, $Q \neq Q'$. However, it is always true that $S = S'$ (see Proposition 8.3 in [Xia02]).

Let $\mathscr{T}_1$ be the set of all boundary based JTs of W, S_1 be the set of links in any such JT, and $\mathscr{Q}_1$ be the union of leaf cluster sets over all such JTs (one Q per JT). Then, after $HTBS$ starts from a leader agent, the first agent A_i that runs $DoDFT$ and passes the test in line 1 must have its boundary $W_i \in \mathscr{Q}_1$. Suppose A_i sends the first inter-agent $StartNewDFT$ to A_j. Hence, the sender-receiver relation $\langle A_i, A_j \rangle$ identifies the link $\langle W_i, W_j \rangle$ in S_1.

When A_j responds to $StartNewDFT$, W_i is eliminated and W_j is updated (as Y_j, Proc. 7, line 5). The reduced boundary set W', without W_i and W_j but with Y_j, is a well-defined boundary set. Hence, $\mathscr{T}_2$, S_2, and $\mathscr{Q}_2$ can be defined accordingly from W', where each $T \in \mathscr{T}_2$ has one cluster less than $\mathscr{T}_1$ and $S_2 = S_1 \setminus \{\langle W_i, W_j \rangle\}$ (one link less).

Applying the argument for $\mathscr{T}_1$ similarly to $\mathscr{T}_2$, another $StartNewDFT$ will be sent, resulting in $\mathscr{T}_3$, S_3, and $\mathscr{Q}_3$. Since $|W| = \eta$ and $|S_1| = \eta - 1$, after $\eta - 1$ $StartNewDFT$ messages, the reduced boundary set has $|W'| = 1$ and all links in S_1 are identified, which specifies one of the JTs in $\mathscr{T}_1$. □

Acknowledgement

We thank anonymous reviewers for their very helpful comments. Financial support through Discovery Grant from NSERC, Canada is acknowledged.

References

[Awe87] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proc. 19th ACM Symp. Theory of Computing*, pages 230–240, 1987.

[BM10] I. Brito and P. Meseguer. Cluster tree elimination for distributed constraint optimization with quality guarantees. *Fundamenta Informaticae*, 102(3-4):263–286, 2010.

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, CA, 2003.

[DMS$^+$08] P. Doshi, T. Matsui, M. Silaghi, M. Yokoo, and M. Zanker. Distributed private constraint optimization. In *Proc. Inter. Conf. Web Intelligence and Intelligent Agent Technology*, pages 277–281, 2008.

[FLP08] B. Faltings, T. Leaute, and A. Petcu. Privacy guarantees through distributed constraint satisfaction. In *Proc. IEEE/WIC/ACM Intelligent Agent Technology*, pages 350–358, 2008.

[FM95] M. Faloutsos and M. Molle. Optimal distributed algorithm for minimum spanning trees revisited. In *Proc. 14th Annual ACM Symp. Principles of Distributed Computing*, pages 231–237, 1995.

[GD01] P.J. Gmytrasiewicz and E.H. Durfee. Rational communication in multi-agent environments. *Auto. Agents and Multi-Agent Systems*, 4(3):233–272, 2001.

[GHS83] R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Programming Languages and Systems*, 5(1):66–77, 1983.

[GKP98] J. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302316, 1998.

[GPBT06] R. Greenstadt, J.P. Pearce, E. Bowring, and M. Tambe. Experimental analysis of privacy loss in DCOP algorithms. In *Proc. Inter. Joint Conf. Autonomous Agents and Multiagent Systems*, pages 1424–1426, 2006.

[Jen88] F.V. Jensen. Junction tree and decomposable hypergraphs. Technical report, JUDEX, Aalborg, Denmark, February 1988.

[KM01] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *Proc. 17th Inter. Joint Conf. on Artificial Intelligence*, pages 1027–1034, 2001.

[KP98] S. Kutten and D. Peleg. Fast distributed construction of smallk-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[KP08] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20(6):391402, 2008.

[LOF10] T. Leaute, B. Ottens, and B. Faltings. Ensuring privacy through distributed computation in multiple-depot vehicle routing problems. In *Proc. ECAI Workshop on Artificial Intelligence and Logistics*, pages 25–30, 2010.

[MB04] A. Maestre and C. Bessiere. Improving asynchronous backtracking for dealing with complex local problems. In *Proc. 16th European Conf. on Artificial Intelligence*, pages 206–210, 2004.

[MPB$^+$06] R.T. Maheswaran, J.P. Pearce, E. Bowring, P. Varakantham, and M. Tambe. Privacy loss in distributed constraint reasoning: a quantitative framework for analysis and its applications. *J. Autonomous Agents and Multi-Agent Systems*, 13(1):27–60, 2006.

[MSTY05] P.J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligences*, 161(1-2):149–180, 2005.

[MSY08] Marius C. M.C. Silaghi and Makoto Yokoo. ADOPT-ing: unifying asynchronous distributed optimization with asynchronous backtracking. *J. Autonomous Agents and Multi-Agent Systems*, 19(2):89–123, 2008.

[MTB$^+$05] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, and P. Varakantham. Taking DCOP to the realworld: efficient complete solutions for distributed multi-event scheduling. In D. Kudenko, D. Kazakov, and E. Alonso, editors, *Proc. 2004 Inter. Conf. on Autonomous Agents and Multiagent Systems, LNCS (LNAI) 3394*, pages 310–317. Springer, Heidelberg, 2005.

[NCKB12] S. Nobari, T.T. Cao, P. Karras, and S. Bressan. Scalable parallel minimum spanning forest computation. In *Proc. 17th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pages 205–214, 2012.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[PF05] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc. 19th Inter. Joint Conf. on Artificial Intelligence*, pages 266–271, 2005.

[PGM05] M. Paskin, C. Guestrin, and J. McFadden. A robust architecture for distributed inference in sensor networks. In *Proc. Information Processing in Sensor Networks*, pages 55–62, 2005.

[Pri57] R.C. Prim. Shortest connection networks and some generalizations. *Bell Syst. Tech. J.*, (36):1389–1401, 1957.

[SAZB05] M.C. Silaghi, A. Abhyankar, M. Zanker, and R. Bartak. Desk-mates (stable matching) with privacy of preferences, and a new distributed CSP framework. In *Proc. Inter. Florida Artificial Intelligence Research Society Conf.*, pages 83–96, 2005.

[SDMY08] M.C. Silaghi, P. Doshi, T. Matsui, and M. Yokoo. Distributed private constraint optimization problem: cost of privacy loss. In *Proc. AAMAS Workshop on Distributed Constraint Reasoning*, pages 1–12, 2008.

[SF05] M.C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, 161(1-2):25–54, 2005.

[SFP06] M.C. Silaghi, B. Faltings, and A. Petcu. Secure combinatorial optimization simulating DFS tree-based variable elimination. In *Proc. 9th Inter. Symp. on AI and Math.*, pages 1–10, 2006.

[SR04] M.C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *Proc. 3rd Inter. Joint Conf. Autonomous Agents and Multiagent Systems*, pages 1396–1397, 2004.

[SSHF00] M.C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proc. AAAI'2000*, pages 917–922, 2000.

[VKV02] M. Valtorta, Y.G. Kim, and J. Vomlel. Soft evidential update for probabilistic multiagent systems. *Int. J. Approximate Reasoning*, 29(1):71–106, 2002.

[VRAC10] M. Vinyals, J.A. Rodriguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *J. Autonomous Agents and Multi-Agent Systems*, 22(3):439–464, 2010.

[XCD04] Yang Xiang, Junjiang Chen, and Abhi Deshmukh. A decision-theoretic graphical model for collaborative design on supply chains. In A.Y. Tawfik and S.D. Goodwin, editors, *Advances in Artificial Intelligence, LNAI 3060*, pages 355–369. Springer, 2004.

[XH11] Yang Xiang and Frank Hanshar. Multiagent expedition with graphical models. *Inter. J. Uncertainty, Fuzziness and Knowledge-Based Systems*, 19(6):939–976, 2011.

[Xia96] Yang Xiang. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence*, 87(1-2):295–342, 1996.

[Xia02] Yang Xiang. *Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach*. Cambridge University Press, Cambridge, UK, 2002.

[Xia08] Yang Xiang. Building intelligent sensor networks with multiagent graphical models. In N. Ichalkaranje G.P. Wren and L.C. Jain, editors, *Intelligent Decision Making: An AI-Based Approach*, pages 289–320. Springer-Verlag, 2008.

[XMZ14] Yang Xiang, Younis Mohamed, and Wanling Zhang. Distributed constraint satisfaction with multiply sectioned constraint networks. *International J. Information and Decision Sciences*, 6(2):127–152, 2014.

[XPE$^+$93] Yang Xiang, B. Pant, A. Eisen, Michael Beddoes, and David Poole. Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 5:293–314, 1993.

[XS13a] Yang Xiang and Kamala Srinivasan. Boundary set based existence recognition and construction of hypertree agent organization. In O.R. Zaiane and S. Zilles, editors, *Advances in Artificial Intelligence, LNAI 7884*, pages 187–198. Springer-Verlag Berlin Heidelberg, 2013.

[XS13b] Yang Xiang and Kamala Srinivasan. Construction of privacy preserving hypertree agent organization as distributed maximum spanning tree. In O.R. Zaiane and S. Zilles, editors, *Advances in Artificial Intelligence, LNAI 7884*, pages 199–210. Springer-Verlag Berlin Heidelberg, 2013.

[XZ07] Yang Xiang and Wanling Zhang. Multiagent constraint satisfaction with multiply sectioned constraint networks. In Z. Kobti and D. Wu, editors, *Advances in Artificial Intelligence, LNAI 4509*, pages 228–240. Springer-Verlag, 2007.

[XZ08] Yang Xiang and Wanling Zhang. Distributed university timetabling with multiply sectioned constraint networks. In *Proc. 21th Inter. Florida Artificial Intelligence Research Society Conf.*, pages 567–572, 2008.

[Yok01] M. Yokoo. *Distributed Constraint Satisfaction* . Springer, 2001.

[YSH05] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: reaching agreement without revealing private information. *Artificial Intelligence*, (161):229–246, 2005.

[ZJSF08] M. Zanker, D. Jannach, M.C. Silaghi, and G. Friedrich. A distributed generative CSP framework for multi-site product configuration. In M. Klusch, M. Pechoucek, and A. Polleres, editors, *Cooperative Information Agents XII*, pages 131–146. 2008.