Construction of Privacy Preserving Hypertree Agent Organization as Distributed Maximum Spanning Tree

Yang Xiang and Kamala Srinivasan

School of Computer Science, University of Guelph, Canada

Abstract. Decentralized probabilistic reasoning, constraint reasoning, and decision theoretic reasoning are some of the essential tasks of a multiagent system (MAS). Many frameworks exist for these tasks, and a number of them organize agents into a junction tree (JT). Although these frameworks all reap benefits of communication efficiency and inferential soundness from the JT organization, their potential capacity on agent privacy has not been realized fully. The contribution of this work is a general approach to construct the JT organization through a maximum spanning tree (MST), and a new distributed MST algorithm, that preserve agent privacy on private variables, shared variables and agent identities.

1 Introduction

Decentralized probabilistic reasoning, constraint reasoning, and decision theoretic reasoning (referred to below as decision making) are some of the essential tasks of a multiagent system (MAS). Many frameworks exist for these tasks, e.g., AEBN [19] and MSBN [22] for multiagent probabilistic reasoning, ABT [12], ADOPT [13], DPOP [16], Action-GDL [20], DCTE [2], and MSCN [25] for multiagent constraint reasoning, and RMM [7], MAID [10], and CDN [24] for multiagent decision making. Some frameworks do not assume specific agent organizational structure, e.g., AEBN and MAID. Some assume a total order among agents, e.g., ABT. Some use a pseudo-tree organization, e.g., ADOPT and DPOP. Some depend on a junction tree (JT) organization, e.g., Action-GDL, DCTE, MSBN, MSCN, and CDN. The organization is known as *hypertree* in the literature on MSBN, MSCN and CDN, and we refer to JT and hypertree interchangeably.

This work focuses on JT-based agent organizations. Potential advantages of JT organizations include communication efficiency, inferential soundness, and agent privacy. As MAS research progresses, agent privacy has received more attention in recent years [18,4]. However, few studies are known on the privacy issue related to JT-based agent organization (see [22,25] for example). Although existing JT-based MAS frameworks all reap benefits of communication efficiency and inferential soundness, the potential capacity of JT-based organization on agent privacy has not been fully realized.

A framework component critical to agent privacy is construction of the JT-based organization. Some frameworks paid no attention to agent privacy at all during the JT organization construction, and others preserved agent privacy to a limited degree. The main contribution of this work is a new algorithm suite that constructs JT-based agent organizations distributively while preserving agent privacy. To the best of our knowledge, no known JT-based MAS frameworks provide the same degree of agent privacy

O. Zaïane and S. Zilles (Eds.): Canadian AI 2013, LNAI 7884, pp. 199–210, 2013.

enabled by the proposed algorithm, except an alternative approach that we report in a related work [26].

In the next section, we introduce background on JT-based organization and the related privacy issue. In the subsequent section, we describe our approach to JT construction as distributed construction of a maximum spanning tree (MST). After reviewing relevant work on distributed MSTs, we present our new algorithm suite, followed by an analysis of its soundness, complexity, and privacy.

2 JT-Based Multiagent Organization and Agent Privacy

JT-based organizations are used in a number of multiagent frameworks, e.g., [21,23,27,20,2]. Under these frameworks, the application environment is represented by a set of variables, referred to as the *env*. The env is decomposed into a set of overlapping *subenvs*, each being a subset of env. The subenvs are one-to-one mapped to agents of the MAS. Fig. 1 (a) shows the subenv decomposition of a trivial env, where subenv V_i is mapped to agent A_i.

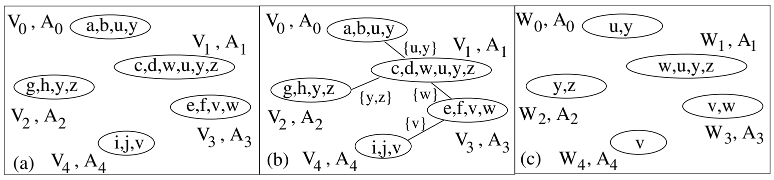

Fig. 1. (a) Subenvs with variables shown in each oval, and their agent association; (b) JT organization with links labeled by shared variables; (c) Agent boundaries.

The subenvs (and hence the agents) are organized into a JT. A JT is a tree where each node is associated with a set called a *cluster*. The tree is so structured that, for any two clusters, their intersection is contained in every cluster on the path between the two (*running intersection*). In a JT-based organization, each cluster corresponds to a subenv and hence an agent. The organization prescribes direct communication pathways between agents. That is, an agent can send a message to another agent, iff they are adjacent in the JT organization.

For each pair of adjacent agents in the JT organization, the intersection of their subenvs is non-empty, and it is used to label the link between corresponding clusters. We refer to the intersection as the set of variables *shared* by the two agents. Fig. 1 (b) shows a JT organization.

Messages between adjacent agents are restricted to be about shared variables only (which is sufficient). In multiagent probabilistic reasoning, a message is the sending agent's belief over shared variables, e.g., in MSBN [21,22]. In multiagent constraint reasoning, a message contains partial solutions over shared variables, e.g., in MSCN [27,25]. In multiagent decision making, a message is either an expected utility function over shared variables, or a partial action plan over them, e.g., in CDN [23,24].

MAS frameworks using JT-based organizations can be grouped based on whether subenvs are assumed *simple* or *complex*. Subenvs are assumed simple under a framework, if each subenv is treated as a subset of variables without internal structure. Subenvs are assumed complex under a framework, if dependence structure within each subenv is explicitly represented and manipulated during multiagent probabilistic reasoning, constraint reasoning, or decision making. Under this criterion, subenvs in Action-GDL [20] and DCTE [2] are simple, while subenvs in MSBN, MSCN, and CDN are complex. In particular, each subenv in a MSBN is modeled as a Bayesian subnet, each subenv in a MSCN is encoded as a constraint subnet, and each subenv in a CDN is represented as a decision subnet.

JT-based agent organizations enable a number of advantages.

Efficient Communication. For agents to cooperate using relevant information local in other agents, it is sufficient to pass two messages along each link of the JT. Hence, time complexity of communication is linear in the number of agents, which is derived from the tree topology of JT.

Soundness of Inference. Global consistency is guaranteed by local consistency, which is derived from running intersection of JT. For instance, multiagent probabilistic reasoning in MSBN [22] is exact. Multiagent decision making in CDN [24] is globally optimal.

Agent Privacy. Most information about individual agents can be kept private while the MAS is fully functioning. This is derived from the restriction of message content in JT-based communication.

Agent privacy above is desirable, when each agent represents an independent principal. For instance, agents collaborating in industrial design with a CDN may represent independent manufacturers in a supply chain [23]. The subenv associated with an agent, modeled as a decision subnet, contains proprietary technical know-hows of a manufacturer, whose non-disclosure is desirable.

More precisely, in a JT-based organization, certain information at individual agents does not need to be exchanged during normal inference, and thus can potentially be kept private. Such information includes the following.

Information Related to a Private Variable. An env variable $x \in V$ is *private*, if it is contained in a single subenv V_i, and hence is associated with a single agent A_i. The information includes its existence, identity, domain (of possible values), associated conditional or marginal probability distribution (in the case of probabilistic reasoning), or constraint (in the cases of constraint reasoning and decision making), and observed or assigned value. In Fig. 1 (a), h is a private variable of A_2.

Information Related to a Shared Variable. An env variable $x \in V$ is *shared*, if it is contained in two or more subenvs, and hence associated with two or more agents. The information about x, as listed above, does not need to be released beyond the agents who share x. In Fig. 1 (a), y is a variable shared by A_0, A_1 and A_2.

Existence and Identity of a Non-bordering Agent. Two agents are *non-bordering* if their subenvs have no common variables. They will not be adjacent in any JT organization, and need not communicate during inference. Hence, they do not need to know the existence or identity of each other. In Fig. 1 (b), A_0 and A_4 are non-bordering.

In summary, agent privacy in a JT-based organization involves at least three types: privacy of private variables, privacy of shared variables, and privacy of agent identity. Although agent privacy is relevant to both MAS frameworks over complex subenvs and those over simple subenvs, it is particularly important for frameworks with complex subenvs, as a simple subenv may contain only a few variables while a complex subenv may include tens or hundreds or more private variables.

For any given JT-based MAS framework, a critical component related to privacy is the construction of JT organization. This is because, once the JT organization is constructed and functioning, it is relatively easy to maintain privacy during normal inference operations, due to message content restriction. It is through the JT construction component, JT-based MAS frameworks demonstrate different degrees of attention to the privacy issue.

In Action-GDL [20], the JT is built from a pseudo-tree, where each node corresponds to one env variable, through a centralized mapping operation. The centralized mapping operation necessarily discloses identities of all variables to the agent who performs the mapping.

The JT used by DCTE [2] is constructed by a method from [15], where agents are initially organized into a tree topology. Each agent starts with a cluster of variables determined by application-based conditions. Hence, the cluster tree generally does not satisfy running intersection property and is not a JT. The cluster tree is transformed into a JT through a message passing process, during which each agent communicates, to each adjacent agent, its local variables as well as variables reachable from other adjacent agents. The message passing thus propagates identities of variables well beyond the agent initially associated with them.

For MSBN, MSCN and CDN frameworks, the JT-based organization is constructed by a centralized coordinator agent [22,25], who has the knowledge of variables shared by any pair of agents, but does not know any private variable. Hence, privacy of private variables is ensured. However, shared variables and identities of non-bordering agents are disclosed to the coordinator, and the corresponding types of privacy are compromised. In this work, we develop a new distributed JT algorithm that respects all three types of privacy.

3 Distributed JT Construction as MST

Consider a MAS consisting of a set $\mathscr{A} = \{A_0,...,A_{\eta-1}\}$ of $\eta > 1$ agents. Let V be the set of env variables, and be decomposed into a collection of *subenvs*, $\Omega = \{V_0,...,V_{\eta-1}\}$, such that $\cup_{i=0}^{\eta-1} V_i = V$. Agents in $\mathscr{A}$ are one-to-one mapped into subenvs in Ω. If A_i and A_j have shared variables, i.e., $V_i \cap V_j \neq \emptyset$, we refer to the set of shared variables, $I_{ij} = V_i \cap V_j$ $(i \neq j)$, as their *border*. A JT-based agent organization is a cluster tree, where each cluster is a subenv and each link is labeled by a border, such that running intersection property holds. Fig. 1 (b) shows a JT organization, where the border between A_1 and A_2 is $I_{12} = \{y,z\}$.

In [22], it is observed that all private variables of a subenv in the above representation can be congregated into a single private variable. If a JT can be constructed using the congregated subenvs, by replacing the congregated private variable with the original private variables, the new cluster tree is still a valid JT.

In this work, we go one step further and observe that the congregated private variable is not needed. For each agent A_i, we denote the set $W_i = \cup_{j \neq i} I_{ij}$ as its *boundary*. We refer to $W = \{W_0, ..., W_{\eta-1}\}$ as the *boundary set* of the MAS. Fig. 1 (c) illustrates agent boundaries. The following Proposition establishes the fact that JT-based organization can be investigated without reference to private variables, whose proof is straightforward.

Proposition 1. *Let V be env of a MAS, Ω be its subenv decomposition, W be the boundary set, and T be a JT with boundaries in W as clusters. Let T' be a cluster tree with subenvs of Ω as clusters, such that it is isomorphic to T with each subenv cluster mapped to the corresponding boundary cluster in T. Then T' is a JT.*

Based on Prop. 1, we can construct a JT-based organization based on the boundary set. Note that this task formulation on JT organization construction immediately guarantees privacy of private variables, as they have been excluded from the specification of the task.

Given the boundary set of an MAS, a JT may or may not exist. Figure 2 (a) shows a boundary set whose elements cannot be organized into a JT.

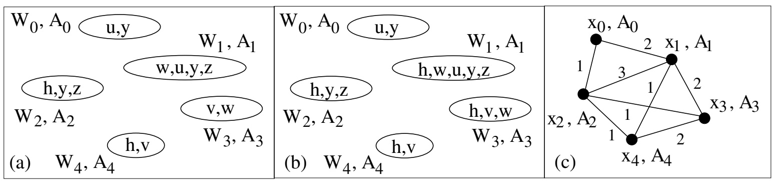

Fig. 2. (a) A boundary set that does not admit JT organization; (b) A boundary set that has a JT organization; (c) Weighted graph defined from the boundary set in (b)

In this work, we assume that there exists a JT for the given boundary set, and we focus on how to distributively construct a JT organization from the boundary set while preserving agent privacy. In a related work [26], we consider how to identify the existence of a JT organization distributively given a boundary set without disclosing agent privacy.

The task of distributed construction of JT-based agent organization can now be stated as follows.

- A set $\mathscr{A} = \{A_0, ..., A_{\eta-1}\}$ of agents is associated with the boundary set $W = \{W_0, ..., W_{\eta-1}\}$, such that a JT exists, with elements of W as its clusters.
- Agents A_i and A_j know the identity of each other and can exchange messages, iff they have a border $I_{ij} = W_i \cap W_j \neq \emptyset$.

– Agent A_i knows only $W_i = \cup_{j \neq i} I_{ij}$, and nothing about variables in other boundaries beyond W_i.

The task of agents is to compute a JT organization with boundaries as clusters, such that each agent knows its adjacent agents in the JT, and the process does not disclose information on agent boundaries beyond the initial knowledge state. To solve this problem, we explore the relation between JT and maximum spanning tree (MST) as follows.

– From a boundary set W, define a weighted graph Ψ. For each $W_i \in W$, create a node x_i. Add a link $\langle x_i, x_j \rangle$, iff there is a border between W_i and W_j, and the weight of the link is $w(x_i, x_j) = |I_{ij}|$.
– Let Ψ' be any MST of Ψ. Define a cluster tree T from Ψ', such that for each link $\langle x_i, x_j \rangle$ of Ψ', W_i and W_j are adjacent in T.
– Then T is a JT, iff a JT with elements of W as its clusters exists [8,22].

Fig. 2 (c) illustrates Ψ defined from a boundary set in (b). Note that the above formulation of JT organization as a MST immediately guarantees privacy of shared variables, since the input to MST computation contains only the number of shared variables between each pair of bordering agents, with other information about these variables excluded.

From the relation between JT and MST, the task of distributed construction of JT organization can be cast as follows. Let the weighted graph Ψ be defined distributively, namely, each agent A_i is associated with node x_i, and knows A_j and $w(x_i, x_j)$ iff $\langle x_i, x_j \rangle \in \Psi$. Compute a MST Ψ' by passing messages only between adjacent agents in Ψ, such that each agent knows its adjacent agent in Ψ', and the process does not disclose information on agent identity and weight association beyond the initial agent knowledge state.

4 Work Related to Distributed MST Construction

Before presenting our privacy preserving distributed MST algorithm, we review relevant literature. Since minimum or maximum spanning trees differ only in the comparison operator (*Min* versus *Max*), without confusion, we refer to all of them by MST.

In the pioneering work by Gallager et al. [5], MST fragments (each initially made of a single node) are combined into larger ones according to a level control, until a single fragment is formed. It has a time complexity $O(\eta \; log \; \eta)$. The basic algorithm assumes distinct link weights, which generally does not hold in our application. To accommodate nondistinct link weights, the modified algorithm either appends node identities to link weights or identifies fragments by node identities. Since link weights and fragment identifies are propagated through messages, node identities will be disclosed beyond node adjacency.

Awerbuch [1] proposed a three-stage algorithm, which was later improved by [3], with time complexity $O(\eta)$. It starts with a counting stage to get η. Then the algorithm in [5] is run to grow each fragment to an $\Omega(\eta / log \; \eta)$ size. A variant of [5] follows, with a more accurate level updating to speed up computation. Non-distinct link weights are handled using the same technique as [5], appending node identities to link weights. Hence, the method suffers from the same node identity disclosure as [5].

An improved algorithm with time complexity $O(d + \eta^{0.613}\ log*\ \eta)$ is proposed in [6], where d is diameter (maximum length of a simple path) of the weighted graph. It first uses a variant of [5] to produce multiple fragments of small diameters, and then combines them into a MST by a rooted operation. Its limitation on node identity disclosure is identical to [5] and [1].

In [11], a two-part algorithm is developed. 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 these fragments into a MST by the same rooted operation as [6]. Since the first part employs a simplified version of [5], its limitation on node identity disclosure is identical to the above algorithms.

An approximate MST algorithm is presented in [9]. Due to the necessary and sufficient condition between JT and MST, an approximate MST cannot yield a JT organization, and hence the method is not applicable to our task.

Recently, an algorithm for computing a set of MSTs, one for each component of a disconnected graph, was proposed [14]. As a parallel algorithm, access of the entire graph by each processor is assumed, and hence it is applicable only when privacy is not a concern at all.

In summary, existing work on distributed MST construction has largely ignored the issue of node identity disclosure.

5 Distributed MST Construction with Privacy of Agent Identity

In this section, we present a new distributed algorithm for MST construction that does not disclose node identity to non-adjacent nodes. In the context of JT organization construction, this translates into non-disclosure of agent identity to non-bordering agents. For this purpose of privacy preservation, we take a different approach than [5] and its extensions [1,3,6,11]. Rather than growing multiple fragments simultaneously, we extend Prim's algorithm [17] distributively and grow a MST through a rooted control. As the result, our algorithm does not assume distinct link weights and needs not to append node identities to link weights either. As will be shown below, our algorithm ensures that no node identity is disclosed beyond adjacency.

Precisely stated, the task is as follows. Given a distributed representation of a connected, weighted graph Ψ of η nodes, construct a MST T by distributed computation. As each node is associated with an agent, without confusion, we refer to node and agent interchangeably. Distributed Ψ representation means that each node has the initial knowledge about each adjacent node (called *neighbor* or *nb*) and the link weight. It knows nothing about the existence of other nodes nor the links and weights between them. For any node v and a nb x of v, the weight of link $\langle v,x\rangle$ is $w(v,x)$.

The algorithm initializes T with an arbitrary node, referred to as the *root* and builds T up as a *directed*, single-rooted tree in $\eta - 1$ rounds. An *outgoing* link of T is a link of Ψ with only one end in T. In each round, a best outgoing link $\langle p,c\rangle$ is selected, where p is in T, and c is added to T. We refer to p as the *tree-parent* of c, and c as a *tree-child* of p. For any node in T, we refer to its tree-parent or a tree-child as its *tree-nb*.

In addition to the initial knowledge, each node v maintains the following local data structure.

1. The state of v is indicated by variable $state \in \{OUT, IN, DONE\}$. Value OUT means that v is not yet in T. IN means that v is in T, but not yet finished its computation. $DONE$ means that v is in T and has finished its computation.
2. Knowledge of v on the state of each nb x is maintained by variable $nbstate(x) \in \{OUT, IN, DONE\}$.
3. The *tree-parent* of v in T is indicated by a pointer so named.
4. A *best outgoing weight table* (BOWT) is maintained. Each row is indexed by a nb x of v, that may lead to outgoing links, and contains the best weight of these links known to v, denoted by $bw(x)$.

During MST computation, nbs of Ψ exchange four types of messages.

Announce Sender announces to each nb, that the former is in MST.
Expand A tree-parent instructs a tree-child to expand current MST, by finding a new node to add.
Notify A tree-leaf notifies a nb that the latter is added to current MST.
Report A tree-child sends to its tree-parent, either to report its termination, or to report the best outgoing weight via the tree-child, through an argument.

We assume that transmission of each message takes at most one time unit.

When the algorithm starts, every node v in Ψ runs *Init* to initialize local data structure.

Procedure 1 (Init)
```
1  $state = OUT$, tree-parent pointer = null;
2  for each nb x, $nbstate(x) = OUT$;
3  create BOWT with one row per nb;
4  for each row of BOWT indexed by x, $bw(x) = w(v,x)$;
```

An arbitrary node is elected as the *root*. It starts the MST computation by executing *Start*. It first adds itself to T, and then runs *Expand* to expand T.

Procedure 2 (Start)
```
1  $state = IN$;
2  send Announce message to each nb;
3  run Expand;
```

Proc. *Expand* can either be called (as in *Start*), or run in response to an *Expand* message. The node selects a nb y that leads to a best outgoing link, adds y to T if y is OUT, otherwise asks y to expand T.

Procedure 3 (Expand)
```
1  select nb $y = \arg\max_x bw(x)$ from BOWT table, breaking ties randomly;
2  if $nbstate(y) = OUT$,
3      send Notify message to y;
4      $nbstate(y) = IN$; record y as a tree-child;
5  else send Expand message to y; // y must be IN
```

When node *v* receives *Notify* message from nb *p*, it is in *T*. It runs Proc. 4 in response. In the process, it announces its status in *T* to its nbs. At the end of the process, *v* replies to *p* with a *Report* message, which contains one of two possible arguments, the *state* value of *v*, or the best outgoing weight *bow*.

Procedure 4 (Response to Notify)

```
1  nbstate(p) = IN; point tree-parent pointer to p;
2  delete the row indexed by p from BOWT;
3  state = IN;
4  for each nb $x \neq p$, send Announce message to x;
5  run Inform;
```

A node *v* runs *Inform* to send *Report* message to its tree-parent *p*.

Procedure 5 (Inform)

```
1  if no nb y with nbstate(y) = OUT
           and each tree-child c has nbstate(c) = DONE,
2      state = DONE;
3      if v is not root, send p message Report(state = DONE);
4  else if v is not root,
5      compute maxbow = $\max_x bw(x)$ from BOWT;
6      send p message Report(bow = maxbow);
```

When a node *v* receives *Announce* message from a nb *x*, it performs the following. Note that root cannot receive *Announce* from its tree-child, but can receive from its non-child tree-descendent.

Procedure 6 (Response to Announce)

```
1  nbstate(x) = IN; delete the row indexed by x from BOWT;
2  if state $\neq$ OUT, run Inform;
```

When a tree-parent *v* receives *Report* from a tree-child *c*, it performs Proc. 7. The report allows *v* to know whether *c* has terminated and, if not, to update its knowledge on the best outgoing link weight through *c*.

Procedure 7 (Response to Report)

```
1  if message argument is (state = DONE),
2      nbstate(c) = DONE; delete the row indexed by c from BOWT;
3      if no nb y with nbstate(y) = OUT
               and each tree-child c has nbstate(c) = DONE,
4          state = DONE;
5          if v is not root, send tree-parent p message Report(state = DONE);
6          else return; // root termination
7  else if maxbow $\neq$ bw(c) in BOWT, // argument is (bow = maxbow)
8      bw(c) = maxbow;
9  if v is not root,
10     compute maxbow' = $\max_x bw(x)$ from BOWT;
11     send tree-parent p message Report(bow = maxbow');
12 else if no pending Report messages, run Expand; // v is root
```

When Proc. 7 returns from line 6 at root, the algorithm suite halts. Consider the example in Fig. 2 (b). Suppose x_0 is the root, whose *BOWT* at start is $(x_1 : 2;\ x_2 : 1)$. It sends *Announce* to x_2 and *Notify* to x_1. Node x_1 removes x_0 from its *BOWT*, sends *Announce* to x_2, x_3 and x_4, and $Report(bow = 3)$ to x_0. Based on the report, x_0 revises its *BOWT* to $(x_1 : 3;\ x_2 : 1)$, and sends *Expand* to x_1.

BOWT at x_1 is $(x_2 : 3;\ x_3 : 2;\ x_4 : 1)$. Hence, x_1 sends *Notify* to x_2, which replies with $Report(bow = 1)$. Based on the report, x_1 sends $Report(bow = 2)$ to x_0. After x_0 has processed *Announce* from x_2 and $Report(bow = 2)$ from x_1, its *BOWT* is $(x_1 : 2)$. Hence, x_0 sends *Expand* to x_1, which in turn sends *Notify* to x_3.

Eventually, x_4 is notified by x_3 and sets its state to *DONE*. When x_2 receives *Announce* from x_4, it sets its state to *DONE* as well. Node x_3 sets $state = DONE$ when it receives *Report* from x_4, and x_1 does so upon receiving *Report* from x_3. In the end, x_0 receives *Report* from x_1 and terminates the computation.

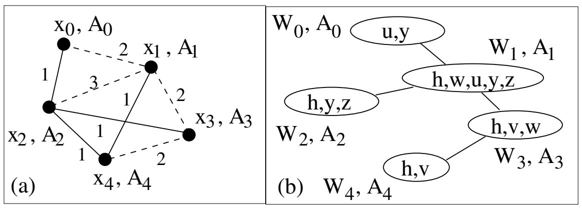

Fig. 3. (a) The MST (dashed links) from Fig. 2 (c); (b) The JT organization.

The resultant MST is shown in Fig. 3 (a) by dashed links, and the corresponding JT organization is in (b). Throughout the computation, no agent identity is communicated.

6 Soundness and Complexity

We refer to the algorithm suite as DPMST, whose soundness is established below.

Proposition 2. *Given a connected, weighted graph Ψ, DPMST computes a MST T of Ψ that is specified distributively, such that each node knows its tree-nbs.*

Proof: DPMST will compute a MST because it extends Prim's algorithm distributively. The recursive executions of Proc. 7 let the root know where an outgoing link with the best weight is located, and recursive executions of Proc. 3 add the other end of the link to T.

When a node v is added to T, it knows its notifier as its tree-parent, and its notifier knows v as its tree-child. Hence, when DSMSTC halts, each node knows its tree-nbs in T. □

We analyze the 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.

Communication cost: Each node is added to T 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 it is added to T: a subtotal of $O(e)$ messages. After a node is added to T, *Report* messages are propagated to the root from the node (Proc. 4) as well as its nbs (Proc. 6): $O(r\ d)$ messages. This yields a subtotal of $O(r\ d\ \eta)$ messages. Hence, the total number of messages is $O(r\ d\ \eta + e)$.

Time complexity: The $O(d\ \eta)$ $Notify/Expand$ messages are sequential, and take $O(d\ \eta)$ time. *Announce* messages by the same sender take $O(r)$ time. The $O(2e)$ *Announce* messages take $O(r\ \eta)$ time. The $O(r\ d)$ *Report* messages due to one node addition form r parallel sequences and take $O(d)$ time. The $O(r\ d\ \eta)$ *Report* messages take $O(d\ \eta)$ time. Hence, time complexity is $O((d+r)\ \eta)$.

Agent privacy: By using the boundary set of a MAS, privacy of private variables is preserved. By using the weighted graph defined from the boundary set, privacy of shared variables is preserved. Since our distributed MST algorithm does not disclose node identity, privacy of agent identity is preserved.

7 Conclusion

Our contribution is a general approach for JT organization construction based on MST, and an algorithm suite for MST construction. Combination of our approach and algorithm suite guarantees agent privacy on private variables, shared variables, as well as agent identity. To the best of our knowledge, no existing JT-based MAS frameworks enable agent privacy at such a degree, except a related work based on boundary set elimination which we report in [26].

The method proposed here assumes that a JT organization exists for the given env decomposition. Whether the condition (JT existence) holds is not dealt with in the current work, and is detected distributively in [26].

Our distributed MST algorithm is efficient, but not as efficient as the most efficient existing algorithms, although they do not allow preservation of agent identity. An open question is whether it is possible for a distributed MST algorithm to be as efficient as these algorithms while preserving agent identity.

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

References

1. Awerbuch, B.: Proc. 19th ACM Symp. Theory of Computing, pp. 230–240 (1987)
2. Brito, I., Meseguer, P.: Cluster tree elimination for distributed constraint optimization with quality guarantees. Fundamenta Informaticae 102(3-4), 263–286 (2010)
3. Faloutsos, M., Molle, M.: Optimal distributed algorithm for minimum spanning trees revisited. In: Proc. 14th Annual ACM Symp. Principles of Distributed Computing, pp. 231–237 (1995)
4. Faltings, B., Leaute, T., Petcu, A.: Privacy guarantees through distributed constraint satisfaction. In: Proc. IEEE/WIC/ACM Intelligent Agent Technology, pp. 350–358 (2008)
5. Gallager, R., Humblet, P., Spira, P.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Programming Languages and Systems 5(1), 66–77 (1983)

6. Garay, J., Kutten, S., Peleg, D.: A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM J. Comput. 27(1), 302–316 (1998)
7. Gmytrasiewicz, P., Durfee, E.: Rational communication in multi-agent environments. Auto. Agents and Multi-Agent Systems 4(3), 233–272 (2001)
8. Jensen, F.: Junction tree and decomposable hypergraphs. Tech. rep., JUDEX, Aalborg, Denmark (February 1988)
9. Khan, M., Pandurangan, G.: A fast distributed approximation algorithm for minimum spanning trees. Distributed Computing 20(6), 391–402 (2008)
10. Koller, D., Milch, B.: Multi-agent influence diagrams for representing and solving games. In: Proc. 17th Inter. Joint Conf. on Artificial Intelligence, pp. 1027–1034 (2001)
11. Kutten, S., Peleg, D.: Fast distributed construction of smallk-dominating sets and applications. J. Algorithms 28(1), 40–66 (1998)
12. Maestre, A., Bessiere, C.: Improving asynchronous backtracking for dealing with complex local problems. In: Proc. 16th European Conf. on Artificial Intelligence, pp. 206–210 (2004)
13. Modi, P., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligences 161(1-2), 149–180 (2005)
14. Nobari, S., Cao, T., Karras, P., Bressan, S.: Scalable parallel minimum spanning forest computation. In: Proc. 17th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, pp. 205–214 (2012)
15. Paskin, M., Guestrin, C., McFadden, J.: A robust architecture for distributed inference in sensor networks. In: Proc. Information Processing in Sensor Networks, pp. 55–62 (2005)
16. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proc. 19th Inter. Joint Conf. on Artificial Intelligence, pp. 266–271 (2005)
17. Prim, R.: Shortest connection networks and some generalizations. Bell Syst. Tech. J. (36), 1389–1401 (1957)
18. Silaghi, M., Abhyankar, A., Zanker, M., Bartak, R.: Desk-mates (stable matching) with privacy of preferences, and a new distributed CSP framework. In: Proc. Inter. Florida Artificial Intelligence Research Society Conf., pp. 83–96 (2005)
19. Valtorta, M., Kim, Y., Vomlel, J.: Soft evidential update for probabilistic multiagent systems. Int. J. Approximate Reasoning 29(1), 71–106 (2002)
20. Vinyals, M., Rodriguez-Aguilar, J., Cerquides, J.: 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)
21. Xiang, Y.: A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. Artificial Intelligence 87(1-2), 295–342 (1996)
22. Xiang, Y.: Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach. Cambridge University Press, Cambridge (2002)
23. Xiang, Y., Chen, J., Deshmukht, A.: A decision-theoretic graphical model for collaborative design on supply chains. In: Tawfik, A.Y., Goodwin, S.D. (eds.) Canadian AI 2004. LNCS (LNAI), vol. 3060, pp. 355–369. Springer, Heidelberg (2004)
24. Xiang, Y., Hanshar, F.: Multiagent expedition with graphical models. Inter. J. Uncertainty, Fuzziness and Knowledge-Based Systems 19(6), 939–976 (2011)
25. Xiang, Y., Mohamed, Y., Zhang, W.: Distributed constraint satisfaction with multiply sectioned constraint networks. Accepted to appear in International J. Information and Decision Sciences (2013)
26. Xiang, Y., Srinivasan, K.: Boundary set based existence recognition and construction of hypertree agent organization. In: ZaÏane, O., Zilles, S. (eds.) AI 2013. LNCS (LNAI), vol. 7884, Springer, Heidelberg (2013)
27. Xiang, Y., Zhang, W.: Multiagent constraint satisfaction with multiply sectioned constraint networks. In: Kobti, Z., Wu, D. (eds.) Canadian AI 2007. LNCS (LNAI), vol. 4509, pp. 228–240. Springer, Heidelberg (2007)