

---

---

# Distributed Constraint Satisfaction with Multiply Sectioned Constraint Networks

Yang Xiang, Younis Mohamed and Wanling Zhang  
Univ. of Guelph, Canada

**Abstract** We propose a new algorithmic framework, multiply sectioned constraint networks (MSCNs), for solving distributed constraint satisfaction problems (DisCSPs) with complex local problems. An MSCN is converted into a linked junction forest (LJF) and is solved by a complete algorithm. Its time complexity is linear on the number and size of local problems (each in charge by an agent) and is exponential on cluster size of LJF. We show that the MSCN-LJF algorithm is more efficient than junction tree-based DisCSP algorithms. When a DisCSP is not naturally an MSCN, we show how to convert it into an MSCN, so that any DisCSP can be solved as above.

## 1 Introduction

A broad range of complex decision problems can be solved as DisCSPs, including sensor network coordination [Bejar et al(2005)], transportation vehicle scheduling [Calisti and Neagu(2004)], and meeting scheduling [Wallace and Freuder(2005)]. Algorithms solving DisCSPs can be classified broadly as being based on distributed backtracking (e.g., ABT [Maestre and Bessiere(2004), Silaghi and Faltings(2005), Bessiere et al(2005)], AFC [Meisels and Zivan(2007)], ADOPT [Modi et al(2005)]), on distributed iterative improvement (e.g., DBA [Hirayama and Yokoo(2005)], DSA [Zhang et al(2005)]), and on dynamic programming (e.g., DPOP [Petcu and Faltings(2005)]). Since every DisCSP can be solved as a distributed constraint optimization problem (DisCOP) [Modi et al(2005)], instances of DisCOP algorithms (e.g., ADOPT and DPOP) are also included above. Some algorithms (e.g., DSA) do not depend on specific agent organization. Others assume a total order among them (e.g., ABT and AFC). Still others use a pseudo-tree (e.g., ADOPT and DPOP) or junction tree (JT) organization (e.g., [Vinyals et al(2010), Brito and Meseguer(2010)]). Many algorithms assume a single variable per agent in their typical formulations (e.g., DSA, ADOPT, DPOP). Complex local problems are being addressed in recent years (e.g., [Maestre and Bessiere(2004), Ezzahir et al(2007), Burke(2008)]).

JTs have long been applied to solving centralized CSPs as in [Dechter and Pearl(1988)] and [Dechter and Pearl(1989)], as well as to centralized probabilistic reasoning with Bayesian

---

networks (see, e.g., [Jensen and Nielsen(2007)]). Subsequently, LJFs are developed as runtime agent organization for multiagent probabilistic reasoning with multiply sectioned Bayesian networks (MSBNs) [Xiang et al(1993), Xiang(2002), Xiang and Hanshar(2010)]. Although JT-based DisCSP algorithms have been proposed in recent years (e.g., in [Vinyals et al(2010)] and [Brito and Meseguer(2010)]), LJFs have never been explored for solving DisCSPs.

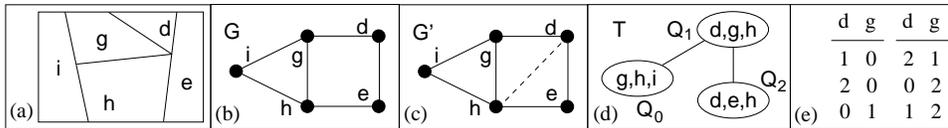
In this work, we show that LJF-based message passing can be applied to solving DisCSPs with complex local problems. A LJF has a JT organization of agents, just as in JT-based DisCSP algorithms. However, local variables in each agent are organized into a single cluster in JT-based DisCSP algorithms. With LJF, they are organized into a local JT, which allows much refined decomposition of local problem and more efficient local problem solving. Furthermore, interface between adjacent agents in JT-based DisCSP algorithms is a single cluster separator. With LJF, the interface is also organized into a JT, which allows interface decomposition and more efficient inter-agent message passing.

Remainder of the paper is organized as follows: Section 2 defines DisCSPs and Section 3 defines MSCNs, a sub-class of DisCSPs, which are directly solvable by LJF-based message passing. In Section 4, we present an alternative formulation of JT-based message passing for solving CSPs to facilitate development of our MSCN algorithm. LJF representation of MSCN is presented in Section 5 and its properties are analyzed. Our algorithm to solve MSCNs based on LJFs is presented in Sections 6 and 7, as well as its completeness and complexity. Section 8 addresses construction of agent organization for MSCNs. Section 9 shows how to convert any DisCSP into an MSCN. Proofs are available at first author’s website.

## 2 Problem Definitions

### 2.1 CSP

A constraint network (CN) is a pair  $\mathcal{R} = (V, \Lambda)$ .  $V \neq \emptyset$  is a set of discrete variables, which we refer to as the *env* (environment). Each variable  $v \in V$  has a finite *domain*  $D_v \neq \emptyset$ , the set of possible values of  $v$ . For any subset  $X \subseteq V$ , its *space*  $D_X$  is the Cartesian product of domains of variables in  $X$ . Each  $\bar{x} \in D_X$  is a *config* (configuration) of  $X$ .  $\Lambda \neq \emptyset$  is a set of constraints. Each *constraint* is a relation  $R_X \subseteq D_X$ , where  $X \subset V$  is the *scope* of the constraint. When a constraint involves a *universal* relation  $U_X = D_X$ , we refer to it as a *dumb* constraint (imposing no restriction). The union of scopes of all constraints covers env, i.e.,  $\cup_{R_X \in \Lambda} X = V$ .



**Fig. 1** (a) A map coloring CN; (b) primal graph; (c) triangulated graph; (d) resultant JT; (e) constraint  $d \neq g$ .

**Example 1** A map coloring CN is shown in Fig. 1 (a), where each region may be colored by red, green or blue, such that adjacent regions differ in color.

Its env is  $V = \{d, e, g, h, i\}$ , where each variable represents the color of a region and has domain  $\{\text{red}, \text{green}, \text{blue}\}$ , which we simply write as  $\{0, 1, 2\}$ . The constraint set is  $\Lambda = \{d \neq g, d \neq e, e \neq h, g \neq h, g \neq i, h \neq i\}$ . Constraint  $d \neq g$  of scope  $\{d, g\}$  is the relation  $R_{\{d,g\}}$ :

$$\{(d = 1, g = 0), (d = 2, g = 0), (d = 0, g = 1), (d = 2, g = 1), (d = 0, g = 2), (d = 1, g = 2)\}.$$

It can also be expressed as the table in Fig. 1 (e), or as

$$R_{\{d,g\}} \text{ over } (d, g) = \{(1, 0), (2, 0), \dots\}.$$

Denote *projection* of config  $\bar{x}$  to  $Y \subseteq X$  by  $\pi_Y(\bar{x})$ . For instance,  $\bar{x} = (d = 0, e = 2, g = 1)$  is a config of  $X = \{d, e, g\}$ . Its projection to  $Y = \{e, g\}$  is the config  $\bar{y} = (e = 2, g = 1)$ . Denote the projection of relation  $R_X$  to  $Y \subseteq X$  by  $\pi_Y(R_X)$ , which consists of the projection of each config in  $R_X$  to  $Y$ . A config  $\bar{x} \in D_X$  *satisfies* constraint  $R_Y$  if either  $X \cap Y = \emptyset$  ( $R_Y$  is irrelevant) or  $\pi_{X \cap Y}(\bar{x}) \in \pi_{X \cap Y}(R_Y)$  (the projection of  $\bar{x}$  to  $X \cap Y$  matches the projection of one config in  $R_Y$ ). A config  $\bar{x}$  is *legal* if it satisfies every constraint in  $\Lambda$ . A *solution* to CN  $\mathcal{R}$  is a legal config over  $V$ . A CSP involves finding a solution for a CN.

## 2.2 Constraint Graphs

Constraints of  $\mathcal{R} = (V, \Lambda)$  can be depicted by a *primal graph*  $G = (V, E)$ , where each node is labeled by a variable  $v \in V$  and an undirected link  $\langle u, v \rangle \in E$  if there exists  $R_X \in \Lambda$  such that  $u \in X$  and  $v \in X$ . Note that primal graphs thus defined depict both binary and higher-order constraints. The primal graph for the above CN is shown in Fig. 1 (b).

A CN  $\mathcal{R}$  can be solved using a structure converted from its primal graph  $G$ . A *cluster*  $C$  is a subset of  $V$ . A *cluster tree* connects a set of clusters into a tree, where each link, called a *separator*, connects two clusters with a non-empty intersection  $S \neq \emptyset$  and is labeled by  $S$ . A cluster tree  $T$  is a JT if the intersection of every two clusters is contained in every separator on the path between them (the *running intersection* property).  $T$  is a JT of a given graph  $G$  if, for each cluster  $C$  of  $T$ , elements of  $C$  are pairwise connected in  $G$ , and no superset  $C' \supset C$  has this property ( $C$  is maximal). Conversion of an arbitrary graph into a JT consists of triangulation, cluster identification, and JT construction outlined below:

A graph is *triangulated* if every cycle of length greater than 3 has two nonadjacent nodes connected by a link.  $G$  in Fig. 1 (b) is not triangulated. A graph  $G$  can be triangulated by node elimination. A node in  $G$  is *eliminated* if its adjacent nodes are pairwise connected (by adding links, called *fill-ins*, if necessary), and the node is deleted as well as links incident to it. After all nodes are eliminated, add all fill-ins produced in the process to the original  $G$ . The resultant graph is triangulated. Fig. 1 (b) is triangulated into (c) by eliminating nodes in the order  $(i, e, d, g, h)$  and adding the dashed link as a fill-in.

A given graph  $G$  has a JT iff  $G$  is triangulated. After  $G$  in (b) is triangulated into (c), each cluster of nodes in (c) that is maximally pairwise connected is identified. There are three of them as shown in (d). They are connected into JT  $T$  in (d). See [Xiang(2002), Dechter(2003)] for more details on JT construction. We write  $C \in T$  if  $C$  is a cluster in  $T$ . We refer to  $G$  and  $T$  as constraint graphs associated with  $\mathcal{R}$ .

### 2.3 DisCSP

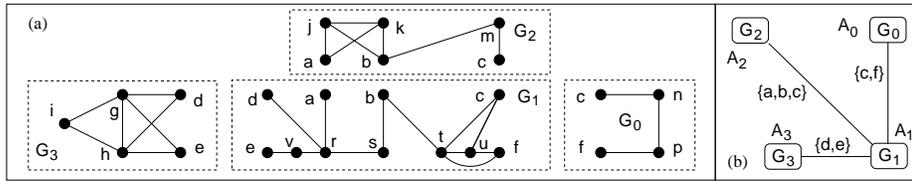
A distributed constraint network (DisCN) is a tuple  $\mathcal{R} = (\mathcal{A}, V, \Omega, \Lambda, \Theta)$ .  $\mathcal{A} = \{A_0, \dots, A_{\eta-1}\}$  is a set of  $\eta > 1$  agents. The set  $V$  of env variables are decomposed into a collection of *subenvs*,  $\Omega = \{V_0, \dots, V_{\eta-1}\}$ , such that  $\cup_{i=0}^{\eta-1} V_i = V$ . The set  $\Lambda$  of constraints are decomposed into  $\Theta = \{A_0, \dots, A_{\eta-1}\}$ , where for each constraint  $R_X$  in  $A_i$ ,  $X \subset V_i$  holds. A solution to the DisCN is a legal config over  $V$ . A DisCSP involves finding a solution for a DisCN.

Each agent  $A_i$  is associated with a *local CN*  $\mathcal{R}_i = (V_i, A_i)$ . If  $x \in V$  has a constraint with  $y \in V_i$  and another constraint with  $z \in V_j$ , then  $x \in V_i \cap V_j$ . We refer to  $x$  as a *shared* variable of  $A_i$  and  $A_j$ . We refer to the set of shared variables,  $I_{ij} = V_i \cap V_j$ , as the *border* between  $A_i$  and  $A_j$ .  $I_{ij}$  is known to both agents. Each variable  $y \in V_j \setminus I_{ij}$  is a *private* variable of  $A_j$  (relative to  $A_i$ ).  $A_i$  is assumed to have no knowledge about the identity of  $y$ , its domain, and constraints  $y$  involves, which we refer to as the *agent privacy*.

The above formulation differs from one-variable-per-agent assumption in a number of DisCSP algorithms, and is intended to express DisCSPs where local problems are complex and some variables are private. The remaining operations are intended to preserve agent privacy, i.e., not to disclose the identity, the domain, and participating constraints of every private variable.

A local CN  $\mathcal{R}_i$  can be depicted by a *local primal graph*  $G_i = (V_i, E_i)$ . Consider local primal graphs  $G_i$  and  $G_j$ . We assume that if link  $\langle x, y \rangle \in E_i$  and  $x, y \in V_j$ , then  $\langle x, y \rangle \in E_j$ . That is, constraints between shared variables are identical among agents involved. We refer to the primal graph depicting  $(V, \Lambda)$  the *global primal graph*  $G = (V, E)$ . Each shared variable appears in  $G$  as a single node. Given the above assumption, the subgraph of  $G$  spanned by  $V_i$  is exactly the local primal graph  $G_i$ .

**Example 2** Fig. 2 illustrates a DisCN with four agents. Agent  $A_0$  has subenv  $V_0 = \{c, f, n, p\}$



**Fig. 2** (a) Local primal graphs of a DisCN that is an MSCN, where each link depicts a  $\neq$  constraint. (b) The hypertree of MSCN.

and local primal graph  $G_0$ , as shown in (a). The domain of each variable is  $\{0, 1, 2\}$ . Each constraint is binary (with the scope over two variables). Variable  $c$  is shared between agents  $A_0, A_1$  and  $A_2$ . For  $A_0$ ,  $p$  is a private variable.

A common misunderstanding regards the above DisCSP formulation as departing from the so-called *private variables, shared constraints* (PVSC) convention, and therefore being too restrictive. We show below that there is no such thing as *private variables in shared constraints*, and therefore, our formulation is general.

Suppose  $A_i$  and  $A_j$  in a PVSC DisCSP share a constraint  $R_X$  over  $X = \{x_i, x_j\}$ , where  $x_i \in D_{x_i}$  and  $x_j \in D_{x_j}$ . By view of PVSC,  $x_j$  is private from  $A_i$ . This is a misconception. Indeed, if  $R_X$  covers all values in  $D_{x_j}$ , then  $A_i$  knows  $D_{x_j}$  by sharing  $R_X$ . If  $R_X$  covers only a

subset  $D_{x_j}^- \subset D_{x_j}$ , then any value in  $D_{x_j} \setminus D_{x_j}^-$  cannot be part of the solution.  $D_{x_j}$  can then be equivalently replaced by  $D_{x_j}^-$ , which  $A_i$  knows by sharing  $R_X$ . To keep name of  $x_j$  private from  $A_i$ ,  $x_j$  can always be obfuscated by a codename. Hence, in what name  $x_j$  is known to  $A_i$  does not matter.

In our formulation, both  $x_i$  and  $x_j$  are shared as well  $R_X$ . No more and no less restriction is assumed by each agent, in comparison with PVSC DisCSPs. Therefore, any DisCSP can be expressed in terms of the above formulation.

### 3 Multiply Sectioned Constraint Network

We consider DisCSPs for a sub-class of DisCNs with complex local problems and can be solved effectively by LJF-based message passing. They are termed MSCNs, as their structures are similar to MSBNs [Xiang(2002)]: graphical models for multiagent probabilistic reasoning.

**Definition 1 (MSCN)** A DisCN  $\mathcal{R} = (\mathcal{A}, V, \Omega, \Lambda, \Theta)$  is a MSCN if the following holds:

1. A JT exists with  $\Omega$  as the set of clusters.
2. Each local primal graph is connected.

The JT condition requires an MSCN to satisfy the relevance property: When the JT exists, subenvs in  $\Omega$  can be reordered as  $V'_0, \dots, V'_{n-1}$  such that for each  $i > 1$  there exists  $j < i$  such that  $I_{ij} \neq \emptyset$ . Hence, each subenv  $V_i$  is relevant to solving the DisCSP. If a DisCN does not satisfy relevance, it can be split into two or more MSCNs, each satisfying relevance.

The JT condition also requires an MSCN to satisfy running intersection (Section 2.2). When subenvs  $V_1, V_2, V_3$  form a path  $\langle V_1, V_2, V_3 \rangle$  in a cluster tree, it means that constraints between  $A_1$  and  $A_3$  are mediated through  $A_2$ . The running intersection simply requires that, if  $A_1$  and  $A_3$  share variable  $x$ , then  $x$  should also be shared by  $A_2$ . This condition is important to efficiently solving MSCNs with complex local problems while preserving agent privacy, as will be seen. In Section 9, we consider how to convert DisCNs violating the running intersection into MSCNs.

Assuming the JT condition holds, we consider how to construct the JT in Section 8.2. Once constructed, we refer to the JT as a *hypertree* and each subenv  $V_i$  as a *hypernode*. We associate the hypernode with local CN  $\mathcal{R}_i$ , local primal graph  $G_i$ , and agent  $A_i$ .  $A_i$  and  $A_j$  are *adjacent* if  $V_i$  and  $V_j$  are adjacent in the hypertree, and we refer to their border  $I_{ij}$  as their *agent interface*.

The second condition in Def. 1 is for simplicity. It naturally holds for most DisCSPs with complex local problems. Otherwise, it can be forced by adding dumb constraints.

The DisCN in Fig. 2 is an MSCN. Its hypertree is shown in (b) with agent interfaces labeled. Below, we consider how to solve the DisCSP given an MSCN.

### 4 Solving CSP With JT Representation

Solving CSPs by JT-based message passing is presented in literature [Dechter and Pearl(1988)], [Dechter and Pearl(1989), Dechter(2003)]. We extend the CSP method to the MSCN-based DisCSPs. This section formulates the CSP method alternatively for several reasons: (1) We present as a set of procedures that can be individually called by the MSCN algorithm. (2) Completeness of our formulation is formally justified in a self-contained manner (rather than through

other tree-solving algorithms as in the above references). (3) Necessity of JTs (rather than just any cluster trees) is not explicit in the original formulation, e.g., [Dechter and Pearl(1989)]. In fact, the issue cannot be clarified easily through other tree-solving algorithms. This necessity is highlighted here. (4) The self-contained analysis forms a base to establish completeness of the MSCN-based algorithm presented in later sections.

Given a CN  $\mathcal{R}$ , the set of all solutions is its *solution set*. Prop. 1 establishes an equivalent specification of the solution set, where  $\bowtie$  is the relational operator *natural join*.

**Proposition 1** ( ) *Let  $\mathcal{R} = (V, \Lambda)$  be a CN.*

1. *The solution set of  $\mathcal{R}$  is the relation  $Sol = \bowtie_{R \in \Lambda} R$ .*
2.  *$\mathcal{R}$  has solution iff  $Sol \neq \emptyset$ .*

The CSP method converts the primal graph of  $\mathcal{R}$  into a JT  $T$  (Section 2.2). For every constraint  $R_X \in \Lambda$ , there exists a cluster  $Q$  in  $T$  where  $X \subseteq Q$ . Proc. 1 assigns constraints in  $\mathcal{R}$  to clusters of  $T$ , so that  $T$  retains the equivalent constraint information. Its complexity is  $O(|\Lambda| k^q)$ , where  $k$  binds domain sizes for variables in  $V$  and  $q$  binds sizes of clusters in  $T$ .

**Procedure 1 (AssignConsToJT)**

*Input: A CN  $\mathcal{R} = (V, \Lambda)$  and a JT  $T$  constructed from its primal graph.*

- 1 *for each constraint  $R_X$  in  $\Lambda$ , assign  $R_X$  to a cluster  $Q$  in  $T$  such that  $X \subseteq Q$ ;*
- 2 *for each cluster  $Q$  in  $T$ ,*
- 3 *denote the set of constraints assigned to  $Q$  by  $\Lambda_Q$ ;*
- 4 *replace  $\Lambda_Q$  by a single constraint  $R_Q = U_Q \bowtie (\bowtie_{R \in \Lambda_Q} R)$ ;*
- 5 *associate  $Q$  with a **relation variable**  $v_Q$  whose domain is  $R_Q$ ;*
- 6 *for each pair of adjacent clusters  $Q$  and  $C$  in  $T$  with separator  $S$ ,*
- 7 *denote an element of  $R_Q$  by  $\bar{q}$  and that of  $R_C$  by  $\bar{c}$ ;*
- 8 *assign **project-equal** constraint  $\pi_S(\bar{q}) = \pi_S(\bar{c})$  over  $v_Q$  and  $v_C$ ;*

We refer to  $T$  as the *JT representation* of  $\mathcal{R}$ . Each cluster  $Q$  in  $T$  is associated with a relation  $R_Q$  and a relation variable  $v_Q$  with domain  $R_Q$ . Each separator in  $T$  is associated with a project-equal constraint over two corresponding relation variables. The set of relation variables  $\mathcal{Q} = \{v_Q | Q \in T\}$  and the set  $\Lambda'$  of project-equal constraints specified over pairs of elements in  $\mathcal{Q}$  define a *derived* binary CN  $(\mathcal{Q}, \Lambda')$ .

Prop. 2 states that the solution set of  $(\mathcal{Q}, \Lambda')$  is identical to that of  $\mathcal{R}$ .

**Proposition 2 (Solution Equivalence)** ( ) *Let  $T$  be a JT representation of CN  $\mathcal{R}$  and  $(\mathcal{Q}, \Lambda')$  be the binary CN derived from  $T$ . Let  $Sol$  be the solution set of  $\mathcal{R}$  and  $Sol'$  be the solution set of  $(\mathcal{Q}, \Lambda')$ . Then,  $Sol' = Sol = \bowtie_{Q \in T} R_Q$ , where  $Q$  is any cluster in  $T$ .*

The CSP method then solves  $(\mathcal{Q}, \Lambda')$  based on directional arc-consistency in  $T$ . Given two clusters  $Q$  and  $C$  of  $T$  with  $S = Q \cap C$ , configs  $\bar{q}$  of  $Q$  and  $\bar{c}$  of  $C$  are *consistent* if  $\pi_S(\bar{q}) = \pi_S(\bar{c})$  (agreeing on their common variables).  $Q$  is *consistent relative to  $C$*  where  $Q \cap C \neq \emptyset$  if, for each config in  $R_Q$ , there exists a consistent config in  $R_C$ . This can be written as  $\pi_{Q \cap C}(R_Q) \subseteq \pi_{Q \cap C}(R_C)$ .

Let  $Q^*$  be any cluster in  $T$  and direct  $T$  with  $Q^*$  as the root. Then each two adjacent clusters form a parent-child pair.  $T$  is *locally directional arc-consistent* relative to root  $Q^*$  if for

every pair of clusters  $Q$  and  $C$ , where  $Q$  is the parent of  $C$ ,  $Q$  is consistent relative to  $C$ .  $T$  is *regionally directional arc-consistent* relative to root  $Q^*$  if for every pair of clusters  $Q$  and  $C$ , where  $Q$  is an ancestor of  $C$ ,  $Q$  is consistent relative to  $C$ .

If  $T$  is an arbitrary cluster tree, it can be locally directional arc-consistent while not being regionally directional arc-consistent. As a result, different clusters could choose partial solutions that extend into solutions of adjacent clusters, but these extended partial solutions are inconsistent to each other. Prop. 3 shows that if  $T$  is a JT, locally directional arc-consistency ensures regionally directional arc-consistency.

**Proposition 3 (Regional directional AC)** *Let  $T$  be a JT representation of a CN and be locally directional arc-consistent relative to cluster  $Q^*$ . Then  $T$  is regionally directional arc-consistent relative to  $Q^*$ .*

The CSP method achieves directional arc-consistency by Proc. 2, activated recursively at each cluster in  $T$  by a *caller*. In the first activation, caller is  $T$ . In subsequent activations, caller is an adjacent cluster. After Proc. 2 (called in  $Q^*$  by  $T$ ) terminates,  $T$  is locally directional arc-consistent relative to  $Q^*$ .

**Procedure 2 (CollectSepCons)** *When caller calls in cluster  $Q$ , it acts as follows:*

*$Q$  calls CollectSepCons in each adjacent cluster  $C$  except caller;  
for each cluster  $C$  (whose separator with  $Q$  is  $S$ ),  
     $Q$  receives from  $C$  a constraint  $R_S$ ;  
    if  $R_S = \emptyset$ ,  $Q$  sends  $\emptyset$  to caller and halts;  
     $Q$  assigns  $R_Q = R_Q \bowtie R_S$ ;  
    if  $R_Q = \emptyset$ ,  $Q$  sends  $\emptyset$  to caller and halts;  
if  $R_Q = \emptyset$ ,  $Q$  sends  $\emptyset$  to caller and halts;  
if caller is a cluster (whose separator with  $Q$  is  $S'$ ),  $Q$  sends  $\pi_{S'}(R_Q)$  to caller;  
else  $Q$  returns a special set  $\nabla$  to signify successful completion;*

Complexity of CollectSepCons is  $O(t k^q)$ , where  $t$  is the number of clusters in  $T$  and  $O(k^q)$  is complexity of the join operation. It can be slightly improved [Dechter(2003)]. Prop. 4 shows that CollectSepCons acts correctly according to the solution set of  $(\mathcal{Q}, \mathcal{A}')$ .

**Proposition 4 (No Solution)** *Let  $T$  be a JT representation of CN  $\mathcal{R}$ ,  $(\mathcal{Q}, \mathcal{A}')$  be the binary CN derived from  $T$ , and  $Sol$  be their solution set. Let CollectSepCons be called in a cluster  $Q^*$  in  $T$ . Then,  $Q^*$  returns  $\emptyset$ , iff  $Sol = \emptyset$ .*

After CollectSepCons,  $T$  is locally directional arc-consistent, as shown below:

**Proposition 5 (Local directional AC)** *Let  $\mathcal{R}$  be a CN and its solution set be  $Sol \neq \emptyset$ . Let  $T$  be a JT representation of  $\mathcal{R}$  and CollectSepCons be called in a cluster  $Q^*$  in  $T$ . Then,  $T$  is locally directional arc-consistent relative to  $Q^*$ .*

After CollectSepCons is called in  $Q^*$ , if  $\emptyset$  is returned,  $\mathcal{R}$  has no solution and the CSP method halts. Otherwise,  $\mathcal{R}$  can be solved by  $T$  calling Proc. 3 in  $Q^*$  with a flag *singleton = true*. It will then be called recursively at each cluster.

**Procedure 3 (DistribSepCons)** *When caller calls in cluster  $Q$  with a singleton flag, it does the following:*

if caller is a cluster (whose separator with  $Q$  is  $S$ ),  
 $Q$  receives from caller a constraint  $R_S$ ;  
 $Q$  assigns  $R_Q = R_Q \bowtie R_S$ ;  
if  $singleton = true$ ,  $Q$  removes all configs in  $R_Q$  except one;  
for each adjacent cluster  $C$  (whose separator with  $Q$  is  $S'$ ) except caller,  
 $Q$  calls *DistribSepCons* in  $C$  with  $\pi_{S'}(R_Q)$  and *singleton* flag;

After *DistribSepCons* is called in  $Q^*$ , the solution to  $\mathcal{R}$  can be obtained by retrieving  $R_Q$  from each cluster  $Q$  and joining them. The CSP method halts. Its complexity is dominated by that of *CollectSepCons* and is  $O(t k^q)$ .

*CollectSepCons* above only achieves directional arc-consistency. A parent cluster  $Q$  is consistent relative to a child cluster  $C$ , but  $C$  may not be consistent relative to  $Q$ . This is possible because the constraint  $R_S$  sent from  $C$  to  $Q$  during *CollectSepCons* may contain a config  $\bar{s}$  such that no config  $\bar{q}$  in  $R_Q$  satisfies  $\pi_S(\bar{q}) = \bar{s}$ . Adjacent clusters  $Q$  and  $C$  are *consistent* if  $Q$  is consistent relative to  $C$  and vice versa.  $T$  is *locally fully arc-consistent* if every pair of adjacent clusters is consistent.  $T$  is *regionally fully arc-consistent* if every pair of clusters of a nonempty intersection is consistent. From Prop. 3, we have Corollary 1.

**Corollary 1 (Regional full AC)** *Let  $T$  be a JT representation of a CN and be locally fully arc-consistent. Then  $T$  is regionally fully arc-consistent.*

Full arc-consistency is not needed to solve CNs. However, it is needed for solving MSCNs as will be seen. *DistribSepCons* with the flag *singleton = false* can be performed after *CollectSepCons* to make  $T$  locally fully arc-consistent. Proc. 4 combines *CollectSepCons* and *DistribSepCons*. It renders a JT regionally fully arc-consistent as summarized by Prop. 6. Its complexity is  $O(t k^q)$ .

#### Procedure 4 (UnifyCons)

choose a cluster  $Q^*$  arbitrarily;  
call *CollectSepCons* in  $Q^*$ ;  
if  $Q^*$  returns  $\emptyset$ , return false;  
call *DistribSepCons* in  $Q^*$  with *singleton = false*;  
return true;

**Proposition 6 (Property of UNifyCons)** *Let  $T$  be the JT representation of a CN  $\mathcal{R}$ .*

1.  $\mathcal{R}$  has no solution iff *UnifyCons* returns false.
2. Otherwise, *UnifyCons* returns true and  $T$  is regionally fully arc-consistent.

The above procedures and their formal properties are used below to develop the MSCN algorithm and prove its completeness.

## 5 Linked Junction Forest Representation of MSCN

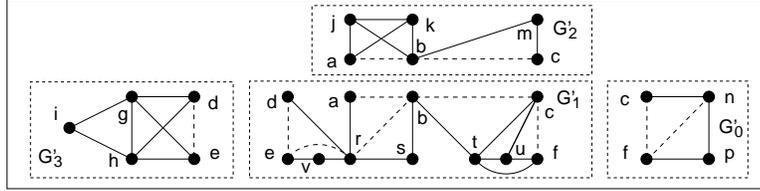
### 5.1 LJF and Its Construction

We extend LJF runtime representation in multiagent probabilistic reasoning [Xiang et al(1993), Xiang(2002)] to solving MSCNs. The idea is to apply JT-based message passing at different

abstract levels. At the lower level, we apply JT-based message passing in each subenv. At the higher level, we apply JT-based message passing to the hypertree. Key to efficiency and privacy preserving lies in seamless integration of the two levels of message passing. LJF provides the structure for such integration.

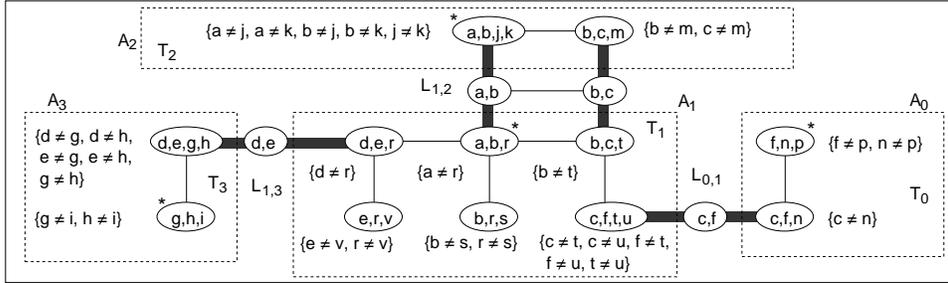
An MSCN is first converted into a LJF. The conversion involves triangulation, local JT construction, and linkage tree (LT) construction. During conversion, the hypertree acts as the agent organization. That is,  $A_i$  communicates directly to  $A_j$ , iff they are adjacent on the hypertree. We illustrate LJF construction with the MSCN in Fig. 2.

**Example 3** *To enable the lower level JT-based message passing, each local CN is converted into a JT representation. First, the global primal graph is triangulated by distributed triangulation, during which agents communicate along hypertree. The communication ensures that fill-ins between shared variables are added consistently at adjacent agents. Each  $G_i$  in Fig. 2 is thus converted to triangulated graph  $G'_i$  in Fig. 3. Then, for each  $G'_i$ , each cluster of nodes maximally*



**Fig. 3** Local primal graphs of MSCN are triangulated. Dashed links between nodes are fill-ins.

*pairwise connected is identified, and these clusters are connected into a local JT  $T_i$  (bounded by box in Fig. 4).*



**Fig. 4** LJF constructed from Fig. 2. Linkage hosts are indicated by thick lines. Constraints assigned to each cluster are in  $\{\}$ .

*To enable seamlessly integration of lower level JT-based message passing with the higher level, each agent interface is converted into a JT representation: the LT. Agent interface between  $A_0$  and  $A_1$  is converted into LT  $L_{0,1}$  with a single cluster. This is a degenerated case due to small size of the example. In such cases, agent interface is handled the same way as JT-based DisCSP algorithms, e.g., [Vinyals et al(2010), Brito and Meseguer(2010)].*

More generally, interface between  $A_1$  and  $A_2$  is converted into LT  $L_{1,2}$  with two clusters. Each cluster in  $L_{1,2}$  is referred to as a **linkage**, e.g.,  $\{b, c\}$ . Each linkage has two **host clusters** one in each JT it links. For instance, linkage  $\{b, c\}$  has host cluster  $\{b, c, t\}$  in  $T_1$  and  $\{b, c, m\}$  in  $T_2$ . The pathway from a host to a linkage, and to the other host bridges two levels of JT-based message passing as will be seen. Although  $L_{1,2}$  contains only two clusters due to small size of the example, for larger subenvs, a LT with many more clusters are possible. Decomposition of agent interface into LT for bridging two levels of message passing allows LJP representation to gain better efficiency than JT-based DisCSP algorithms as we will show.

Graph structures resultant from the conversion, local JTs and LTs, together with the hypertree, will be used to organize JT-based message passing at both levels. Their properties are summarized below:

1. Primal graph of each local CN is converted into a local JT. Hence, JT-based message passing (Section 4) is applicable locally.
2. For each constraint  $R_X$  in each local CN, there exists a cluster  $Q$  in the local JT such that  $X \subseteq Q$ . Hence, constraints in each local CN can be transferred to clusters in the local JT.
3. Let  $X$  be a subset of shared variables in local primal graphs  $G_i$  and  $G_j$ , and  $T_i$  and  $T_j$  be the local JTs, respectively. Then whenever  $X$  is contained in a cluster in  $T_i$ , there exists a cluster  $Q$  in  $T_j$  such that  $X \subseteq Q$ . Hence, constraints over  $X$  can be easily propagated across agents.
4. Each agent interface is converted into a LT that is a JT. Hence, local arc-consistency ensures regional arc-consistency in LTs (see Corollary 1).
5. Only triangulation involves communication and remaining operations are local. All operations preserve agent privacy.

After structural conversion, constraints in each local CN are transferred to the local JT. Each agent  $A_i$  assigns constraints in  $A_i$  to clusters in  $T_i$  by AssignConsToJT (Section 4).

**Example 4** Consider AssignConsToJT by  $A_1$ . Constraints assigned to cluster  $\{c, f, t, u\}$  are shown in Fig. 4. The resultant relation  $R_{\{c, f, t, u\}}$  is the following:

$c$	$f$	$t$	$u$	$c$	$f$	$t$	$u$
0	0	1	2	1	1	2	0
0	0	2	1	2	2	0	1
1	1	0	2	2	2	1	0

The relation variable  $v_{\{c, f, t, u\}}$  has domain  $R_{\{c, f, t, u\}}$ . Similarly, cluster  $\{b, c, t\}$  is associated with relation variable  $v_{\{b, c, t\}}$  with domain  $R_{\{b, c, t\}}$ . Because  $\{c, f, t, u\}$  and  $\{b, c, t\}$  are adjacent clusters, a project-equal constraint is assigned between  $v_{\{c, f, t, u\}}$  and  $v_{\{b, c, t\}}$ . It requires that config that  $v_{\{c, f, t, u\}}$  takes from  $R_{\{c, f, t, u\}}$  and config that  $v_{\{b, c, t\}}$  takes from  $R_{\{b, c, t\}}$  are identical on  $c$  and  $t$ .

Table 1 shows relations of all local JT clusters, where relations of the ‘same’ set of configs are listed only once. For instance, relation over cluster  $\{g, h, i\}$  in  $T_3$  and relation over cluster  $\{b, c, m\}$  in  $T_2$  are shown in the middle, and are referred to as  $R_2$  over  $\{g, h, i\}$  and  $R_2$  over  $\{b, c, m\}$ , respectively.

For LTs, each agent uses Proc. 5 to assign constraints to linkages.

#### Procedure 5 (AssignConsToLT)

Input: Local JT  $T_i$  of  $A_i$  and LTs  $\{L_{i,j}\}$ .

**Table 1** Relations associated with local JT clusters. A single line separates scopes of relations with an identical set of configs, enclosed within a pair of double lines.

$R_1$				$R_2$			$R_3$																		
d	e	g	h	g	h	i	1	0	2	d	e	r	0	2	2	<table border="1"> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>0</td></tr> <tr><td>2</td><td>2</td><td>1</td></tr> </table>	2	1	1	2	2	0	2	2	1
2	1	1																							
2	2	0																							
2	2	1																							
a	b	j	k	b	c	m	1	1	0	a	b	r	1	0	0										
c	f	t	u	e	r	v	1	1	2	b	c	t	1	1	0										
0	0	1	2	b	r	s	1	2	0	c	f	n	1	1	2										
0	0	2	1	f	n	p	2	0	1	0	0	1	1	2	0										
1	1	0	2	0	0	1	2	1	0	0	0	2	1	2	2										
1	1	2	0	0	0	2	2	2	0	0	1	1	2	0	0										
2	2	0	1	0	1	2	2	2	1	0	1	2	2	0	1										
2	2	1	0	0	2	1				0	2	1	2	1	0										

for each LT  $L_{i,j}$  with adjacent agent  $A_j$ ,  
for each linkage  $S$  with host cluster  $Q$  in  $T_i$  and host cluster  $C$  in  $T_j$ ,  
denote an element of  $R_Q$  by  $\bar{q}$  and that of  $R_C$  by  $\bar{c}$ ;  
assign constraint  $\pi_S(\bar{q}) = \pi_S(\bar{c})$  over relation variables  $\{v_Q, v_C\}$ ;

For instance, cluster  $\{b, c, t\}$  in  $T_1$  and cluster  $\{b, c, m\}$  in  $T_2$  are the hosts of linkage  $\{b, c\}$ , and a project-equal constraint is assigned between  $v_{\{b,c,t\}}$  and  $v_{\{b,c,m\}}$ . It requires that configs they take are identical on  $b$  and  $c$ .

Conversion of MSCN  $\mathcal{R} = (\mathcal{A}, V, \Omega, \Lambda, \Theta)$  results in

$$\mathcal{F} = (\mathcal{A}, V, \Omega, H, T, \Delta, L, \Phi),$$

where  $H$  is the MSCN hypertree that forms the agent organization,  $T = \{T_0, \dots, T_{\eta-1}\}$  is a set of local JTs, one per subenv in  $\Omega$ , as the local problem solving structure, and  $L = \{L_{i,j}\}$  is a set of LTs, one per agent interface on  $H$ , as the inter-agent message passing structure.  $\Delta = \{\Delta_0, \dots, \Delta_{\eta-1}\}$  is a collection of constraint sets, one per  $T_i$ , expressing intra-agent constraints. Each  $\Delta_i$  is a set of constraints, one per cluster and one per separator (project-equal) of  $T_i$ .  $\Phi = \{\Phi_{i,j}\}$  is a collection of constraint sets one per linkage tree  $L_{i,j}$ , expressing inter-agent constraints. Each  $\Phi_{i,j}$  is a set of project-equal constraints, one per linkage of  $L_{i,j}$ . We refer to  $\mathcal{F}$  as the LJF representation or simply LJF of the MSCN, that will be used for solving the MSCN.

## 5.2 Properties of LJF

We compare the solution set of an MSCN and that of its LJF. Prop. 7 specifies that the solution set of an MSCN is the natural join of constraints in all local CNs.

**Proposition 7 (MSCN solution)** *Let  $\mathcal{R}$  be an MSCN with a set of local CNs  $\{\mathcal{R}_i = (V_i, \Lambda_i)\}$ . The solution set of  $\mathcal{R}$  is the relation  $Sol = \bowtie_i (\bowtie_{R \in \Lambda_i} R)$ .*

Next, we consider the solution set of a LJF. Denote the set of relation variables associated with clusters in  $T_i$  as  $\mathcal{Q}_i = \{v_Q | Q \in T_i\}$  and the union of such sets as  $\mathcal{Q} = \cup_i \mathcal{Q}_i$ . Denote the set of project-equal constraints associated with  $T_i$  as  $\mathcal{A}'_i$ , the set of project-equal constraints associated with  $L_{i,j}$  as  $\mathcal{A}'_{i,j}$ , and the union of these sets as  $\mathcal{A}' = (\cup_i \mathcal{A}'_i) \cup (\cup_{i,j} \mathcal{A}'_{i,j})$ . Then  $(\mathcal{Q}, \mathcal{A}')$  defines a binary CN derived from LJF  $\mathcal{F}$ . Theorem 1 states that the solution set of  $(\mathcal{Q}, \mathcal{A}')$  is identical to that of  $\mathcal{R}$ .

**Theorem 1** *Let  $\mathcal{R}$  be an MSCN,  $\mathcal{F}$  be its LJF, and  $(\mathcal{Q}, \mathcal{A}')$  be the binary CN derived from  $\mathcal{F}$ . Let  $Sol$  be the solution set of  $\mathcal{R}$  and  $Sol'$  be the solution set of  $(\mathcal{Q}, \mathcal{A}')$ . Then,  $Sol' = Sol$ .*

Construction of LJF is dominated by triangulation and AssignConsToJT. Complexity of triangulation is  $O(\eta g^2 d^2)$  [Xiang(2002)], where  $g$  binds  $|V_i|$  and  $d$  binds the number of variables in a single constraint in  $\mathcal{A}$ . Complexity of AssignConsToJT performed by all agents is  $O(\eta \lambda k^q)$ , where  $\lambda$  binds  $|A_i|$ ,  $k$  binds domain sizes for variables in  $V$ , and  $q$  binds sizes for clusters in local JTs. Hence, the overall complexity of LJF construction is  $O(\eta g^2 d^2 + \eta \lambda k^q)$ . The computation is efficient when  $q$  is small, which occurs if the global primal graph of the MSCN is sparse. Note that the value  $q$  is known after distributed triangulation (Section 5) and before AssignConsToJT is performed.

## 6 Achieving Directional Interface-Consistency in LJF

To solve an MSCN using its LJF, we extend directional arc-consistency to LJF. An agent  $A_i$  is *interface-consistent* relative to adjacent agent  $A_j$  if, for each config  $\bar{v}_i$  of  $V_i$  ( $\bar{v}_i \in \times_{R \in A_i} R$ ), there exists a consistent config of  $V_j$ . Direct the hypertree with any agent  $A^*$  as the root. The LJF is *locally directional interface-consistent* relative to  $A^*$  if, for every two agents  $A_i$  and  $A_j$  where  $A_i$  is the parent of  $A_j$ ,  $A_i$  is interface-consistent relative to  $A_j$ . The LJF is *globally directional interface-consistent* relative to  $A^*$  if, for every two agents  $A_i$  and  $A_j$  where  $A_i$  is the ancestor of  $A_j$ ,  $A_i$  is interface-consistent relative to  $A_j$ .

**Example 5** *Suppose the LJF in Fig. 4 is directed with  $A_0$  being the root. The LJF is locally directional interface-consistent relative to  $A_0$ , if  $A_0$  is interface-consistent relative to  $A_1$ ,  $A_1$  is to  $A_2$ , and  $A_1$  is to  $A_3$ . The LJF is globally directional interface-consistent relative to  $A_0$  if, in addition,  $A_0$  is interface-consistent relative to both  $A_2$  and  $A_3$ .*

When agent organization is an arbitrary tree, the system may be locally directional interface-consistent but not globally directional interface-consistent. As a result, different agents may choose partial solutions for their subenvs that extend into partial solutions of subenvs in adjacent agents, but these partial solutions are inconsistent with each other. In other words, two agents may assign the same shared variable with different values even though the LJF is locally directional interface-consistent. Because the hypertree of LJF is a JT, Prop. 8 shows that locally directional interface-consistency ensures globally directional interface-consistency. It can be proven by generalizing proof for Prop. 3.

**Proposition 8 (Global directional IC)** *Let  $\mathcal{F}$  be a LJF of an MSCN and be locally directional interface-consistent relative to agent  $A^*$ . Then  $\mathcal{F}$  is globally directional interface-consistent relative to  $A^*$ .*

Procs. 6 and 7 achieve locally directional interface-consistency in  $\mathcal{F}$ . Proc. 6 is used by  $A_i$  to update linkage host constraints based on message from adjacent  $A_j$ .

**Procedure 6 (AbsorbIntCons)** When  $A_i$  performs *AbsorbIntCons* relative to  $A_j$  with a set  $\Gamma = \{R_X\}$ , where each  $R_X$  is a constraint over a linkage  $X$  with  $A_j$ ,  $A_i$  does the following:

for each linkage  $C$  with  $A_j$  with linkage host  $Q$  at  $A_i$ ,  
 assign  $R_Q = R_Q \bowtie R_C$ , where  $R_C \in \Gamma$ ;  
 if  $R_Q = \emptyset$ , return false;  
 return true;

Proc. 7 recursively propagates messages inwards along hypertree. Agent executing Proc. 7 is referred to as  $A_0$  with local JT  $T_0$ . Execution is activated by a *caller*, who is either an adjacent agent, denoted by  $A_c$ , or a unique *coordinator* agent. Additional adjacent agents of  $A_0$  are denoted by  $A_1, \dots, A_m$ , if any.

**Procedure 7 (CollectIntCons)** When caller calls  $A_0$  to *CollectIntCons*, it acts as follows:

- 1 for each agent  $A_i$  ( $i = 1, \dots, m$ ),
- 2 call *CollectIntCons* on  $A_i$ ;
- 3 if  $A_i$  returns  $\emptyset$ , return  $\emptyset$ ;
- 4 receive  $\Gamma_i = \{R_C\}$  where  $R_C$  is a constraint over a linkage  $C$  with  $A_i$ ;
- 5 perform *AbsorbIntCons* relative to  $A_i$  with  $\Gamma_i$ ;
- 6 if false is returned, return  $\emptyset$ ;
- 7 perform *UnifyCons* in local JT  $T_0$ ;
- 8 if false is returned, return  $\emptyset$ ;
- 9 if  $A_c$  is an adjacent agent,
- 10 initialize  $\Gamma_c = \emptyset$ ;
- 11 for each linkage  $S$  with  $A_c$  of linkage host  $Q$  at  $A_0$ ,
- 12 compute  $R_S = \pi_S(R_Q)$ ;
- 13 add  $R_S$  to  $\Gamma_c$ ;
- 14 send  $\Gamma_c$  to  $A_c$ ;
- 15 else return a special set  $\nabla$  to coordinator signifying successful completion;

**Example 6** We illustrate *CollectIntCons* using LJF in Fig. 4. Suppose coordinator calls *CollectIntCons* in agent  $A_0$ . In turn,  $A_0$  calls *CollectIntCons* in  $A_1$ , which calls *CollectIntCons* in  $A_2$  and  $A_3$ .

$A_3$  performs *UnifyCons* by calling *CollectSepCons* in cluster, say,  $\{g, h, i\}$ , which in turn calls *CollectSepCons* in cluster  $\{d, e, g, h\}$ . In response,  $\{d, e, g, h\}$  sends relation  $R_4$  (Table 2) over  $\{g, h\}$  to  $\{g, h, i\}$ , which causes modification of the relation at  $\{g, h, i\}$  to  $R_5$  (Table 2).

Next,  $A_3$  calls *DistribSepCons* in  $\{g, h, i\}$ , which in turn calls *DistribSepCons* in  $\{d, e, g, h\}$  with  $R_4$  (Table 2). This results in no change in the relation at  $\{d, e, g, h\}$ . *UnifyCons* at  $A_3$  returns with true.  $T_3$  has cluster relations:  $R_1$  (Table 1) for  $\{d, e, g, h\}$  and  $R_5$  (Table 2) for  $\{g, h, i\}$ . Before completing *CollectIntCons*,  $A_3$  sends  $A_1$  a message containing relation  $R_6$  (Table 2) over linkage  $\{d, e\}$ .

Concurrently with  $A_3$ ,  $A_2$  also performs *UnifyCons* by calling *CollectSepCons* in cluster, say,  $\{a, b, j, k\}$ , followed by calling *DistribSepCons* in  $\{a, b, j, k\}$ . During *CollectSepCons*, the message from  $\{b, c, m\}$  to  $\{a, b, j, k\}$  is a universal relation over  $\{b\}$ , which causes no change in  $\{a, b, j, k\}$ . During *DistribSepCons*, the message from  $\{a, b, j, k\}$  to  $\{b, c, m\}$  is the same

**Table 2** Relations as messages between clusters or newly assigned to clusters.

$R_4$	
b	r
c	t
e	r
f	n
g	h
0	1
0	2
1	0
1	2
2	0
2	1

$R_5$		
b	r	s
e	r	v
f	n	p
g	h	i
0	1	2
0	2	1
1	0	2
1	2	0
2	0	1
2	1	0

$R_6$	
a	b
c	f
d	e
0	0
1	1
2	2

$R_7$		
b	c	t
0	0	1
0	0	2
0	1	2
0	2	1
1	0	2
1	1	0
1	1	2
1	2	0
2	0	1
2	1	0
2	2	0
2	2	1

$R_8$		
a	b	r
c	f	n
d	e	r
0	0	1
0	0	2
1	1	0
1	1	2
2	2	0
2	2	1

universal relation that causes no change in  $\{b, c, m\}$ . *UnifyCons* at  $A_2$  returns with true. Before completing *CollectIntCons*,  $A_2$  sends  $A_1$  a message containing two relations with one over each linkage. The relation over  $\{a, b\}$  is  $R_6$  (Table 2) and that over  $\{b, c\}$  is universal.

After  $A_1$  receives the message from  $A_3$ , it calls *AbsorbIntCons*, which causes relation at linkage host  $\{d, e, r\}$  to be modified into relation  $R_8$  (Table 2). Similarly, after receiving the message from  $A_2$ ,  $A_1$  calls *AbsorbIntCons*. It modifies relation at linkage host  $\{a, b, r\}$  into relation  $R_8$  (Table 2) but relation at linkage host  $\{b, c, t\}$  remains the same as  $R_3$  (Table 1).

Subsequently,  $A_1$  performs *UnifyCons* by calling *CollectSepCons* in cluster, say,  $\{a, b, r\}$ , followed by calling *DistribSepCons*. During *CollectSepCons*, message sent from  $\{e, r, v\}$  to  $\{d, e, r\}$  is a universal relation over  $\{e, r\}$  and hence causes no change to constraint at  $\{d, e, r\}$ . Message sent from  $\{d, e, r\}$  to  $\{a, b, r\}$  is a universal relation over  $\{r\}$ . Message from  $\{b, r, s\}$  to  $\{a, b, r\}$  is a universal relation over  $\{b, r\}$ . Message from  $\{c, f, t, u\}$  to  $\{b, c, t\}$  is  $R_4$  (Table 2) over  $\{c, t\}$  and changes relation at  $\{b, c, t\}$  to  $R_7$  (Table 2). Message from  $\{b, c, t\}$  to  $\{a, b, r\}$  is universal over  $\{b\}$ .

During *DistribSepCons*, message from  $\{a, b, r\}$  to  $\{d, e, r\}$  is a universal relation over  $\{r\}$ . Message from  $\{d, e, r\}$  to  $\{e, r, v\}$  is  $R_4$  (Table 2) over  $\{e, r\}$  and it modifies relation at  $\{e, r, v\}$  to  $R_5$  (Table 2). Message from  $\{a, b, r\}$  to  $\{b, r, s\}$  is  $R_4$  (Table 2) over  $\{b, r\}$  and modifies relation at  $\{b, r, s\}$  to  $R_5$  (Table 2). Message from  $\{a, b, r\}$  to  $\{b, c, t\}$  is a universal relation over  $\{b\}$ . Message from  $\{b, c, t\}$  to  $\{c, f, t, u\}$  is  $R_4$  (Table 2) over  $\{c, t\}$  and causes no change to relation at  $\{c, f, t, u\}$ . *UnifyCons* at  $A_1$  returns with true.  $T_1$  has the following cluster relations:  $R_1$  (Table 1) for  $\{c, f, t, u\}$ ,  $R_7$  (Table 2) for  $\{b, c, t\}$ ,  $R_8$  (Table 2) for  $\{d, e, r\}$  and  $\{a, b, r\}$ ,  $R_5$  (Table 2) for  $\{e, r, v\}$  and  $\{b, r, s\}$ . Before completing *CollectIntCons*,  $A_1$  sends  $A_0$  a message containing relation  $R_6$  (Table 2) over linkage  $\{c, f\}$ .

After  $A_0$  receives the message, it calls *AbsorbIntCons* which replaces constraint at linkage host  $\{c, f, n\}$  by  $R_8$  (Table 2). Afterwards,  $A_0$  performs *UnifyCons* by calling *CollectSepCons* in cluster, say,  $\{f, n, p\}$ , followed by calling *DistribSepCons*. During *CollectSepCons*, message from  $\{c, f, n\}$  to  $\{f, n, p\}$  is  $R_4$  (Table 2) over  $\{f, n\}$ . It modifies relation at  $\{f, n, p\}$  into  $R_5$  (Table 2). During *DistribSepCons*, message from  $\{f, n, p\}$  to  $\{c, f, n\}$  is  $R_4$  (Table 2) over  $\{f, n\}$  and has no effect at  $\{c, f, n\}$ . *UnifyCons* at  $A_0$  returns with true.  $T_0$  is regionally fully

arc-consistent with the following cluster relations:  $R_8$  (Table 2) for  $\{c, f, n\}$  and  $R_5$  (Table 2) for  $\{f, n, p\}$ . As the result,  $A_0$  terminates *CollectIntCons* and returns  $\nabla$ .

Theorem 2 shows that after execution of *CollectIntCons*, the LJF reaches consistency at the local JT level, at the agent interface level, as well as at the agent organization level. These levels of consistency ensure that the MSCN solution can be obtained by efficient propagation of partial solutions among agents, detailed in the next section.

**Theorem 2 (LJF Consistency)** *Let  $\mathcal{F} = (\mathcal{A}, V, \Omega, H, T, \Delta, L, \Phi)$  be a LJF of an MSCN and *CollectIntCons* be called on agent  $A_0 \in \mathcal{A}$ .*

*$\mathcal{F}$  has no solution iff  $A_0$  returns  $\emptyset$ . Otherwise,  $A_0$  returns  $\nabla$  and the following holds:*

1.  $\mathcal{F}$  is globally directional interface-consistent relative to  $A_0$ .
2. Each  $T_i$  is regionally fully arc-consistent.
3. Each linkage tree  $L_{i,j}$  is regionally fully arc-consistent.

## 7 Solving MSCN through LJF

As shown in Theorem 2, if  $A_0$  returns  $\nabla$  at the end of *CollectIntCons*, the MSCN has solution. In this section, we show that, in that case, a solution will be obtained through another round of message passing along the hypertree. The denotation of a calling agent  $A_c$ , the executing agent  $A_0$ , and its other adjacent agents  $A_1, \dots, A_m$ , introduced in Section 6 will be used.

In response to message  $\Gamma$  from  $A_c$ , representing a partial solution over the interface,  $A_0$  executes Proc. 8 (from line 3) to generate a partial solution consistent with  $\Gamma$  for its subenv.

**Procedure 8 (GetLocalSol)** *When agent  $A_0$  performs *GetLocalSol* with  $\Gamma = \{R_X\}$ , where each  $R_X$  is a singleton constraint (consisting of one config) over a linkage  $X$  with  $A_c$ , it does the following:*

- 1 if  $\Gamma = \emptyset$ ,
- 2 call *DistribSepCons* with *singleton = true* in any cluster in  $T_0$ ;
- 3 else
- 4 for each linkage  $S$  with  $A_c$  (whose host cluster is  $Q$ ),
- 5 assign  $R_Q = R_Q \bowtie R_S$ , where  $R_S \in \Gamma$ ;
- 6 call *DistribSepCons* with *singleton = true* in the host of any linkage with  $A_c$ ;

Note that after *DistribSepCons* (lines 2 and 6),  $R_Q$  will be a singleton. Proc. 9 below is executed recursively by agents along the hypertree. It uses Proc. 8 to propagate partial solutions over agent interfaces.

**Procedure 9 (DistribSol)** *When caller calls  $A_0$  to *DistribSol*, it does the following:*

- 1 if caller is an adjacent agent,
- 2 receive  $\Gamma = \{R_X\}$  where each  $R_X$  is a singleton constraint over linkage  $X$  with caller;
- 3 perform *GetLocalSol* with  $\Gamma$ ;
- 4 else perform *GetLocalSol* with  $\emptyset$ ;
- 5 for each agent  $A_i$  ( $i = 1, \dots, m$ ),
- 6 initialize  $\Gamma' = \emptyset$ ;
- 7 for each linkage  $S$  with  $A_i$  (whose host cluster is  $Q$ ), add  $\pi_S(R_Q)$  to  $\Gamma'$ ;
- 8 call *DistribSol* on  $A_i$  with  $\Gamma'$ ;

Algorithm 1 combines procedures introduced above to solve the DisCSP. It is executed by the coordinator.

**Algorithm 1 (SolveDisCSP)**

choose an agent  $A^*$  arbitrarily;  
 call *CollectIntCons* in  $A^*$ ;  
 if  $A^*$  returns  $\emptyset$ , return failure;  
 else, call *DistribSol* in  $A^*$ ;

**Example 7** To illustrate *SolveDisCSP*, suppose coordinator executes by choosing  $A^* = A_0$ . Example 6 illustrated *CollectIntCons*. We continue with call of *DistribSol* in  $A_0$ .  $A_0$  runs *GetLocalSol* by first calling *DistributeSepSolution* at, say,  $\{f, n, p\}$ . This produces partial solution  $R_{11}$  for  $\{f, n, p\}$  first and then  $R_{10}$  (Table 3) for  $\{c, f, n\}$  at  $T_0$ .

**Table 3** Relations generated during *DistribSol*.

	$R_{10}$		$R_{11}$		$R_{12}$		$R_{13}$		$R_{14}$
$R_9$	a b r		b r s		a b		a b j k		
c f t u	b c m		e r v		b c		c f t u		g h i
2 2 0 1	b c t		f n p		c f		d e g h		1 0 2
2 2 1 0	c f n		2 1 0		d e		2 2 1 0		
	d e r		2 2						
	2 2 1								

Next,  $A_0$  calls  $A_1$  to *DistribSol* with message containing relation  $R_{12}$  (Table 3) over  $\{c, f\}$ . In response,  $A_1$  modifies its relation in linkage host  $\{c, f, t, u\}$  to  $R_9$ . It then calls *DistribSepCons* in host  $\{c, f, t, u\}$ . The resultant partial solution at each cluster of  $T_1$  are as follows:  $R_{13}$  over  $\{c, f, t, u\}$ ,  $R_{10}$  over  $\{b, c, t\}$ ,  $\{a, b, r\}$ ,  $R_{11}$  over  $\{b, r, s\}$ ,  $R_{10}$  over  $\{d, e, r\}$ , and  $R_{11}$  over  $\{e, r, v\}$ .

After that,  $A_1$  calls  $A_2$  to *DistribSol* with message containing relations  $R_{12}$  over  $\{a, b\}$  and  $\{b, c\}$ . In response,  $A_2$  generates partial solutions  $R_{13}$  (Table 3) over  $\{a, b, j, k\}$  and  $R_{10}$  over  $\{b, c, m\}$  at  $T_2$ .

Similarly,  $A_1$  calls  $A_3$  to *DistribSol* with message containing relation  $R_{12}$  over  $\{d, e\}$ . In response,  $A_3$  generates partial solutions  $R_{13}$  over  $\{d, e, g, h\}$  and  $R_{14}$  over  $\{g, h, i\}$  at  $T_3$ . *SolveDisCSP* now terminates successfully and natural join of the above partial solutions from all agents is a solution:

$$(a = 2, b = 2, c = 2, d = 2, e = 2, f = 2, g = 1, h = 0, i = 2, j = 1,$$

$$k = 0, m = 1, n = 1, p = 0, r = 1, s = 0, t = 1, u = 0, v = 0).$$

Note that for agent privacy, this join operation is not physically performed.

Theorem 3 below establishes completeness of *SolveDisCSP*.

**Theorem 3** Let  $\mathcal{F} = (\mathcal{A}, V, \Omega, H, T, \Delta, L, \Phi)$  be a LJF of an MSCN and *SolveDisCSP* be executed. Then failure is returned iff  $\mathcal{F}$  has no solution. Otherwise,  $R' = \bowtie_i (\bowtie_{Q \in T_i} R_Q)$  is a singleton such that  $R' \subseteq \text{Sol}$ , where  $\text{Sol}$  is the solution set of the MSCN.

Let  $\eta$  be the number of agents,  $t$  be the maximum number of clusters in a local JT,  $q$  be the maximum size of clusters, and  $k$  bind domain sizes for variables in  $V$ . After CollectIntCons completes, SolveDisCSP is backtrack-free. Hence, computation is dominated by UnifyCons during CollectIntCons. UnifyCons has no more than twice the amount of computation of CollectSepCons, whose complexity is  $O(t k^{2q})$  (Section 4). Therefore, the complexity of SolveDisCSP is  $O(\eta t k^{2q})$ . For experimental implementation and empirical evaluation of SolveDisCSP, see [Mohamed(2011)].

## 8 Hypertree Agent Organization

### 8.1 Identifying Hypertree Existence

The hypertree of an MSCN plays the organizational role for the system. Since not every DisCN satisfies condition (1) of Def. 1, we consider identification of hypertree existence. We assume the existence of a coordinator agent  $Co$  who knows the border between each pair of agents in  $\mathcal{A}$ .  $Co$  knows nothing about private variables of any agent. Under this condition, we propose Algorithm 2 for  $Co$  to determine the existence of a hypertree. The idea is for  $Co$  to create a dependency graph among shared variables, and to determine hypertree existence based on the relation between triangulated graphs and JTs (Section 2.2). For each agent  $A_i$ , we denote set  $W_i = \cup_{j \neq i} I_{ij}$  as its *boundary*. That is,  $W_i$  contains shared variables of  $A_i$  relative to all other agents. We refer to  $\mathcal{W} = \{W_i | i = 0, \dots, \eta - 1\}$  as the *boundary collection* of the DisCN.

#### Algorithm 2 (HasHypertree)

for each agent  $A_i$ ,  $W_i = \text{boundary of } A_i$ ;  
 create graph  $G_b$  with nodes labeled by elements of  $\cup_i W_i$ ;  
 for each  $A_i$ , connect each pair of nodes in  $W_i$ ;  
 if  $G_b$  is not triangulated, return no-hypertree;  
 identify each cluster of nodes maximally pairwise connected;  
 if a cluster  $C$  exists such that  $C \neq W_i$  for each  $i$ , return no-hypertree;  
 return has-hypertree;

We refer to  $G_b$  as the *boundary graph* of the DisCN.

**Example 8** For DisCN in Fig. 2,  $Co$  knows non-empty borders between agents:

$$I_{01} = \{c, f\}, \quad I_{02} = \{c\}, \quad I_{12} = \{a, b, c\}, \quad I_{13} = \{d, e\}.$$

$Co$  derives  $W_0 = \{c, f\}$ ,  $W_1 = \{a, b, c, d, e, f\}$ ,  $W_2 = \{a, b, c\}$ , and  $W_3 = \{d, e\}$ .  $G_b$  is shown in Fig. 5 (a) and has a single cluster. HasHypertree returns has-hypertree.

**Example 9** A DisCN over 5 agents has non-empty borders between agents as follows:

$$I_{01} = \{u, y\}, I_{02} = \{y\}, I_{12} = \{h, y, z\}, I_{13} = \{w\}, I_{14} = \{h\}, I_{24} = \{h\}, I_{34} = \{v\}.$$

$Co$  derives  $W_0 = \{u, y\}$ ,  $W_1 = \{h, w, u, y, z\}$ ,  $W_2 = \{h, y, z\}$ ,  $W_3 = \{v, w\}$ , and  $W_4 = \{h, v\}$ .  $G_b$  is shown in Fig. 5 (b). It is triangulated and has two clusters. One of them,  $\{h, v, w\}$ , does not correspond to any  $W_i$ . Hence, HasHypertree returns no-hypertree.

Prop. 9 establishes soundness of HasHypertree.

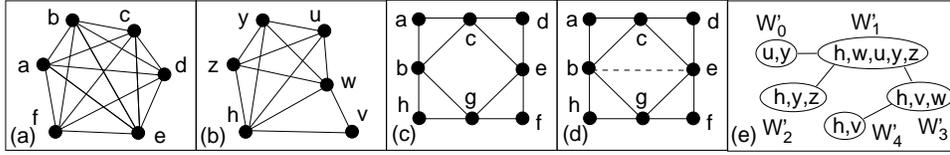


Fig. 5 Graph  $G_b$  for Examples 8 (a), 9 (b), and 11 (c). Triangulated graph (d) and JT (e) for Example 11.

**Proposition 9** *A hypertree exists for a DisCN iff HasHypertree returns has-hypertree.*

One limitation of HasHypertree is that  $Co$  has the knowledge of all shared variables. We expect to relax this requirement in future research.

## 8.2 Construction of Hypertree Agent Organization

Next, we consider construction of hypertree for a given DisCN, assuming that one exists. We assume an integrator agent  $Itr$ , known to each agent in  $\mathcal{A}$ . Recall that each agent  $A_i$  knows each other agent  $A_j$  if they have a non-empty border  $I_{ij} \neq \emptyset$ . We refer to such  $A_j$  as an *associate* of  $A_i$ . We assume that each agent can communicate with its associates. Recall also that adjacent agents refer to those who are adjacent on the hypertree.

To construct hypertree distributively, Algorithm 3 specifies actions by  $Itr$  and each agent. It is based on the well-known maximum spanning tree algorithm for JT construction (see, e.g., [Xiang(2002)]), but extends the method into distributed.

### Algorithm 3 (SetHypertree)

```

each agent registers with  $Itr$ ;
 $Itr$  sets counter  $cnt$  to number of registered agents;
 $Itr$  randomly selects  $A_i$ , notifies  $A_i$  as winner, and sets  $cnt = cnt - 1$ ;
 $A_i$  announces winner status to each associate;
while  $cnt > 0$ ,
   $Itr$  asks each winner to bid for right to select new winner;
  for each winner  $A_i$ ,
    for each non-winner associate  $A_j$ ,  $A_i$  computes  $w_{ij} = |I_{ij}|$ ;
     $A_i$  makes bid  $w_i = \max_j w_{ij}$  and
    denotes corresponding associate as winner candidate;
     $A_i$  sends bid  $w_i$  to  $Itr$ ;
  after receipt of all bids,  $Itr$  selects  $A_k$  such that  $w_k = \max_i w_i$ ;
   $Itr$  notifies  $A_k$  to select new winner;
   $A_k$  notifies its winner candidate, and they denote each other as adjacent;
  the new winner announces its status to each associate and  $Itr$ ;
   $Itr$  sets  $cnt = cnt - 1$ ;
 $Itr$  announces halt and halts;
upon receipt, each agent halts;

```

**Example 10** *Consider the DisCN in Example 8 and Fig. 2. After agent registration, suppose  $Itr$  selects  $A_0$  to be the first winner.  $A_0$  will announce to associates  $A_1$  and  $A_2$ .*

When *Itr* asks  $A_0$  to bid, it bids with  $w_0 = \max(w_{01}, w_{02}) = \max(2, 1) = 2$ , and denotes  $A_1$  as winner candidate. Subsequently, *Itr* notifies  $A_0$  to select,  $A_0$  notifies  $A_1$  to be new winner, and  $A_1$  announces to associates  $A_0$ ,  $A_2$ ,  $A_3$ , and *Itr*.  $A_0$  and  $A_1$  now regard each other as adjacent on hypertree.

Next *Itr* asks  $A_0$  and  $A_1$  to bid.  $A_0$  bids with  $w_0 = 1$  and  $A_1$  bids with  $w_1 = 3$ . Both denote  $A_2$  as candidate. *Itr* notifies  $A_1$ ,  $A_1$  notifies  $A_2$  as the new winner, and  $A_2$  announces to associates  $A_0$ ,  $A_1$ , and *Itr*.  $A_1$  and  $A_2$  now regard each other as adjacent on hypertree.

Afterwards, *Itr* asks  $A_0$ ,  $A_1$ , and  $A_2$  to bid. Bids for  $A_0$ ,  $A_1$  and  $A_2$  are  $w_0 = 0$ ,  $w_1 = 2$ , and  $w_2 = 0$ , and their candidates are null,  $A_3$ , and null, respectively. *Itr* notifies  $A_1$ , and  $A_1$  notifies  $A_3$  as the new winner.  $A_1$  and  $A_3$  now regard each other as adjacent on hypertree. *SetHypertree* then terminates with the hypertree in Fig. 2 (b) constructed.

Prop. 10 shows that after *SetHypertree*, a hypertree emerges for the DisCN.

**Proposition 10** *Let SetHypertree be run in a DisCN where hypertree exists. After it halts, a hypertree is formulated such that each agent knows its adjacent agents in the hypertree.*

From *Itr*'s own notifications and winner announcements, *Itr* can infer the hypertree topology in terms of agent adjacency, as well as the cardinality of each agent interface. *Itr* does not, however, have the knowledge of agent subenvs, nor content of agent interfaces.

## 9 Converting DisCN into MSCN

Next, we consider DisCNs where *HasHypertree* returns no-hypertree. These DisCNs violate Def. 1 and are not MSCNs. *SolveDisCSP* is not applicable to them. We study how to convert them into MSCNs so that *SolveDisCSP* can be applied.

*HasHypertree* returns no-hypertree when boundary graph  $G_b$  is not triangulated, or  $G_b$  is triangulated but has a cluster (or more) that is not contained in any agent boundary (Example 9). To convert such a DisCN into MSCN, we propose to triangulate  $G_b$  (if it is not so) and then enlarge some agent boundaries, such that if *HasHypertree* is applied to the new set of boundaries, has-hypertree will be returned.

Enlargement of some boundaries means inserting *shared* variables that they do not originally contain. This has the potential to disclose those variables, their domains, and constraints they participate to agents who do not originally have such knowledge. To minimize the impact, we assume that newly inserted variables and their domain values will be obfuscated by codenames, as practiced by other DisCSP algorithms, e.g., DPOP [Leaute et al(2010)]. In the following, we focus on triangulation and boundary enlargement.

Our criterion in conversion is to minimize the number of newly inserted shared variables. Suppose *Co* executes *HasHypertree* and finds that  $G_b$  is not triangulated. It can triangulate  $G_b$  into  $G'_b$  by node elimination. Note that no matter how  $G_b$  is triangulated, every boundary  $W_i$  is contained in at least one cluster in  $G'_b$ . From clusters of  $G'_b$ , a new boundary set

$$\mathcal{W}' = \{W'_i | i = 0, \dots, \eta - 1\}$$

is defined.  $\mathcal{W}'$  specifies a new set of subenvs (each obtained by the union of  $W'_i$  and the set of private variables of  $A_i$ ), which can be organized into a hypertree. The smaller the number of fill-ins produced during triangulation, the less number of shared variables will be inserted into  $W'_i$ 's.

Hence, a triangulation with the minimal number of fill-ins is consistent with minimization of newly inserted shared variables. Since optimal triangulation is NP-hard [Yannakakis(1981)], we compromise with a greedy heuristics. To choose the next node to eliminate during triangulation, we apply the min-fill-in heuristic (select the node with the minimum number of fill-ins).

After boundary graph  $G_b$  is triangulated into  $G'_b$ , it is necessary to redefine the boundary for each agent. Example 11 illustrates the technical issue in doing so.

**Example 11 (Boundary)** Consider a DisCN with the following boundary set:

$$\mathcal{W} = \{W_0 = \{a, b, c\}, W_1 = \{c, d, e\}, W_2 = \{e, f, g\}, W_3 = \{b, g, h\}\}.$$

Its boundary graph  $G_b$  is shown in Fig. 5 (c).  $G_b$  is not triangulated and can be triangulated into  $G'_b$  in (d) by adding fill-in  $\langle b, e \rangle$ . However,  $G'_b$  contains two clusters  $\{b, c, e\}$  and  $\{b, e, g\}$ , that do not equal to any  $W_i$ .

The similar happens in Example 9, where cluster  $\{h, w, v\}$  in  $G_b$  does not equal to any  $W_i$ . However,  $\{h, w, v\}$  is a superset of  $W_4$ , and can be assigned to  $A_4$  as its enlarged boundary. Here, neither  $\{b, c, e\}$  nor  $\{b, e, g\}$  is a superset of any  $W_i$ .

We propose Algorithm 4, to be executed by coordinator agent  $Co$ , to redefine (enlarged) agent boundaries. After triangulating  $G_b$  (if it is non-triangulated) into  $G'_b$ , clusters of  $G'_b$  are organized into a JT  $T'$ . If a cluster  $C$  in  $T'$  does not equal to a  $W_i$  nor is a superset of any,  $C$  is merged into an adjacent cluster  $C'$ . This is done recursively until the new cluster  $C'$  is a superset of a  $W_i$ , and it is assigned to  $A_i$  as its enlarged boundary.

**Algorithm 4 (EnlargeBoundary)** Let  $\mathcal{W}$  be the boundary set and  $G_b$  be the boundary graph, such that *HasHypertree* returns no-hypertree.

if  $G_b$  is not triangulated, triangulate it into  $G'_b$ ;

else  $G'_b = G_b$ ;

organize clusters of  $G'_b$  into a JT  $T'$ ;

initialize  $\mathcal{W}'$  to  $\mathcal{W}$ ;

for each cluster  $C$  in  $T'$  that is not a superset of any set in  $\mathcal{W}'$ ,

while  $C$  is not a superset of any set in  $\mathcal{W}'$ ,

merge an adjacent cluster  $C'$  into  $C$  in  $T'$ ;

for each cluster  $C$  in  $T'$  that is not equal to any set in  $\mathcal{W}'$ ,

remove  $W'_i$  from  $\mathcal{W}'$  such that  $|W'_i| = \max_{W_k \subset C} |W_k|$ ;

add cluster  $C$  to  $\mathcal{W}'$  and denote it by  $W'_i$ ;

return  $\mathcal{W}'$ ;

**Example 12** Consider the DisCN in Example 9. Its  $G_b$  is Fig. 5 (b) and is triangulated. Apply Algorithm 4 and  $G'_b = G_b$ . The JT  $T'$  has two clusters  $\{h, u, w, y, z\}$  and  $\{h, v, w\}$ . Initially,

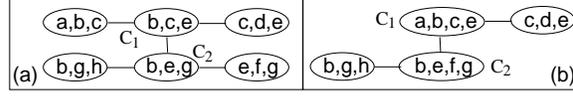
$$\mathcal{W}' = \{W'_0 = \{u, y\}, W'_1 = \{h, w, u, y, z\}, W'_2 = \{h, y, z\}, W'_3 = \{v, w\}, W'_4 = \{h, v\}\}.$$

The first for loop finds no cluster in  $T'$  that satisfies the condition. Cluster  $C = \{h, v, w\}$  is processed by the second for loop. By breaking ties between  $W'_3$  and  $W'_4$  arbitrarily,  $C$  replaces  $W'_3$  in  $\mathcal{W}'$ . The new boundary collection  $\mathcal{W}'$  is

$$\{W'_0 = \{u, y\}, W'_1 = \{h, w, u, y, z\}, W'_2 = \{h, y, z\}, W'_3 = \{h, v, w\}, W'_4 = \{h, v\}\}.$$

Note that a shared variable  $h$  is inserted to boundary  $W'_3$ . A hypertree for the DisCN that is isomorphic to the JT in Fig. 5 (e) can then be constructed.

**Example 13 (More on EnlargeBoundary)** Consider Algorithm 4 applied to the DisCN in Example 11.  $G'_b$  is shown in Fig. 5 (d), the initial JT  $T'$  is shown in Fig. 6 (a), and initially,



**Fig. 6** (a) Initial JT  $T'$  for Example 12. (b) Updated JT  $T'$ .

$$\mathcal{W}' = \{W'_0 = \{a, b, c\}, W'_1 = \{c, d, e\}, W'_2 = \{e, f, g\}, W'_3 = \{b, g, h\}\}.$$

The first for loop must process clusters  $C_1 = \{b, c, e\}$  and  $C_2 = \{b, e, g\}$ . Suppose cluster  $\{a, b, c\}$  is merged into  $C_1$ , and  $\{e, f, g\}$  is merged into  $C_2$ . The updated JT  $T'$  is shown in Fig. 6 (b). The second for loop will replace  $W'_0$  by  $C_1$  and  $W'_2$  by  $C_2$ . The new boundary collection is

$$\mathcal{W}' = \{W'_0 = \{a, b, c, e\}, W'_1 = \{c, d, e\}, W'_2 = \{b, e, f, g\}, W'_3 = \{b, g, h\}\}.$$

A hypertree for the DisCN that is isomorphic to the JT in Fig. 6 (b) can then be constructed. Note that from  $\mathcal{W}'$ , new borders between agents are easily defined:

$$I_{01} = \{c, e\}, I_{02} = \{b, e\}, I_{03} = \{b\}, I_{12} = \{e\}, I_{23} = \{b, g\}.$$

Prop. 11 establishes the key properties of Algorithm 4. Its practical implication is the following: Suppose a DisCN is not an MSCN. If  $Co$  executes HasHypertree, followed by EnlargeBoundary, then the DisCN, modified based on the new boundary collection, is an MSCN.

**Proposition 11 (Make MSCN)** Let  $\mathcal{W}$  be the boundary collection of a DisCN and  $G_b$  be the boundary graph, such that HasHypertree returns no-hypertree. Let  $\mathcal{W}'$  be the new boundary collection returned by applying Algorithm 4 to  $G_b$ . Then the following hold:

1.  $|\mathcal{W}'| = |\mathcal{W}|$ .
2. For each agent  $A_i$ ,  $W'_i \supseteq W_i$ , where  $W'_i \in \mathcal{W}'$  and  $W_i \in \mathcal{W}$ .
3. A JT exists with  $\mathcal{W}'$  as the set of clusters.

In the *while* loop of EnlargeBoundary, an adjacent cluster  $C'$  needs to be selected to merge into  $C$ . Note that the merging is equivalent to adding fill-ins to  $G'_b$ . Hence, the equivalent to min-fill-in heuristic is to prefer  $C'$  that can terminate *while* loop immediately and  $|C' \setminus C|$  is minimal.

Similar to HasHypertree, one limitation of EnlargeBoundary is that  $Co$  has the knowledge of all shared variables. We leave its relaxation to future research.

## 10 Comparison with JT-based Framework

We discuss relation between our MSCN-LJF framework and JT-based framework for solving DisCSPs, e.g., [Vinyals et al(2010), Brito and Meseguer(2010)].

Both frameworks organize subenvs into JTs. In [Brito and Meseguer(2010)], the JT is built as in [Paskin et al(2005)]. Since method in [Paskin et al(2005)] distributes variables among

agents globally, it will disclose private variables. In [Vinyals et al(2010)], the JT is built from a pseudo-tree in a centralized fashion. Since each node in the pseudo-tree corresponds to a variable, it will also disclose private variables. For MSCN-LJF framework, the JT subenv organization is stated in Def. 1 (1). Our methods to build the hypertree (Sections 8 and 9) do not disclose private variables and are able to preserve agent privacy. Our methods require *Co* and *Itr* agents, where *Co* knows all shared variables. These are expected to be relaxed in future research.

Once the JT subenv organization is established, variables in a subenv is treated as a single cluster by the JT-based framework. Each inter-agent message is over a separator of such clusters. For the DisCN in Fig. 2, runtime representation is isomorphic to (b) with each  $G_i$  replaced by cluster  $V_i$ . On the other hand, in the MSCN-LJF framework, variables in each subenv are decomposed into a local JT. Each agent interface is also decomposed into a LT. Not only local inference can be performed at the level of clusters of local JTs, each inter-agent message is over a linkage. The decomposition at both subenv and agent interface levels allows MSCN-LJF framework to be more efficient.

Formally, let  $\eta$  be the number of agents,  $g$  be the maximum number of variables in a subenv, and  $k$  bind domain sizes for variables. Generalizing complexity result of Section 4, time complexity of solving DisCSP in JT-based framework is  $O(\eta k^g)$ . Under MSCN-LJF framework, let  $q$  be the maximum size of clusters in local JTs. Since  $g$  binds number of clusters in local JTs, time complexity of SolveDisCSP is  $O(\eta g k^{2q})$ . As a result, computation time in JT-based framework grows exponentially with the size of subenv. With MSCN-LJF framework, it only grows linearly, when  $q$  value remains the same.

## 11 Conclusion

The contribution of this work is the proposal of a new algorithmic framework, MSCNs, for solving DisCSPs with complex local problems. A MSCN is converted into a LJF based decomposition, and is solved by a complete algorithm. Complexity of the algorithm is linear on the number and size of local problems, and is exponential on cluster size in local JT decomposition. Although not every DisCN is naturally an MSCN, the issue of converting such DisCNs into MSCNs is resolved algorithmically.

Our method differs from existing methods for complex local problems. A number of techniques are proposed in [Burke(2008)] that are intended to be used with any centralized local solver. We present an algorithmic framework where local computation and inter-agent message passing are seamlessly combined and the former directly contributes to efficiency of the latter. Some of the ideas in [Burke(2008)] are implicitly embedded in our framework, e.g., interchangeability. Work in [Maestre and Bessiere(2004), Ezzahir et al(2007)] extends ABT to address complex local problems, while we propose a new algorithmic framework based on LJFs. In comparison with JT-based framework, the MSCN-LJF framework is more efficient and preserves agent privacy.

To identify whether a DisCN is naturally a MSCN and to convert a DisCN into an MSCN, our algorithms require a coordinator agent with access of all shared variables. This requirement is expected to be relaxed through future research. Another direction of future research is to extend the MSCN-LJF framework to DisCOPs.

## References

- [Bejar et al(2005)] Bejar R, Domshlak C, Fernandez C, Gomes C, Krishnamachari B, Selman B, Valls M (2005) Sensor networks and distributed CSP: communication, computation and complexity. *Artificial Intelligence* 161(1-2):117–147
- [Bessiere et al(2005)] Bessiere C, Maestre A, Brito I, Meseguer P (2005) Asynchronous backtracking without adding links: a new member in the ABT family. *Artificial Intelligence* 161(1-2):7–24
- [Brito and Meseguer(2010)] Brito I, Meseguer P (2010) Cluster tree elimination for distributed constraint optimization with quality guarantees. *Fundamenta Informaticae* 102:263–286
- [Burke(2008)] Burke D (2008) Exploiting problem structure in distributed constraint optimization with complex local problems. PhD thesis, U. College Cork, Ireland
- [Calisti and Neagu(2004)] Calisti M, Neagu N (2004) Constraint satisfaction techniques and software agents. In: *Proc. Agents and Constraints Workshop*, pp 1–12
- [Dechter(2003)] Dechter R (2003) *Constraint Processing*. Morgan Kaufmann
- [Dechter and Pearl(1988)] Dechter R, Pearl J (1988) Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence* 34:1–38
- [Dechter and Pearl(1989)] Dechter R, Pearl J (1989) Tree clustering for constraint networks. *Artificial Intelligence* 38(3):353–366
- [Ezzahir et al(2007)] Ezzahir R, Belaissaoui M, Bessiere C, Bouyakhf E (2007) Compilation formulation for asynchronous backtracking with complex local problems. In: *Proc. Inter. Symp. Computational Intelligence and Intelligent Informatics*, pp 205–211
- [Hirayama and Yokoo(2005)] Hirayama K, Yokoo M (2005) The distributed breakout algorithms. *Artificial Intelligence* 161(1-2):89–116
- [Jensen and Nielsen(2007)] Jensen F, Nielsen T (2007) *Bayesian Networks and Decision Graphs* (2nd Ed.). Springer
- [Leaute et al(2010)] Leaute T, Ottens B, Faltings B (2010) Ensuring privacy through distributed computation in multiple-depot vehicle routing problems. In: *Proc. ECAI Workshop on Artificial Intelligence and Logistics*, pp 25–30
- [Maestre and Bessiere(2004)] Maestre A, Bessiere C (2004) Improving asynchronous backtracking for dealing with complex local problems. In: *Proc. 16th European Conf. on Artificial Intelligence*, pp 206–210
- [Meisels and Zivan(2007)] Meisels A, Zivan R (2007) Asynchronous forward-checking for DisCSPs. *Constraints* 12(1):131–150
- [Modi et al(2005)] Modi P, Shen W, Tambe M, Yokoo M (2005) Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1-2):149–180
- [Mohamed(2011)] Mohamed Y (2011) An empirical study of distributed constraint satisfaction algorithms. Master’s thesis, University of Guelph
- [Paskin et al(2005)] Paskin M, Guestrin C, McFadden J (2005) A robust architecture for distributed inference in sensor networks. In: *Proc. Information Processing in Sensor Networks*, pp 55–62
- [Petcu and Faltings(2005)] Petcu A, Faltings B (2005) A scalable method for multiagent constraint optimization. In: *Proc. 19th Inter. Joint Conf. on Artificial Intelligence*, pp 266–271
- [Silaghi and Faltings(2005)] Silaghi M, Faltings B (2005) Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence* 161(1-2):25–54
- [Vinyals et al(2010)] Vinyals M, Rodriguez-Aguilar J, Cerquides J (2010) 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
- [Wallace and Freuder(2005)] Wallace R, Freuder E (2005) Constraint-based reasoning and privacy-efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence* 161(1-2):209–227
- [Xiang(2002)] Xiang Y (2002) *Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach*. Cambridge University Press, Cambridge, UK
- [Xiang and Hanshar(2010)] Xiang Y, Hanshar F (2010) Comparison of tightly and loosely coupled decision paradigms in multiagent expedition. *International J Approximate Reasoning* 51:600–613
- [Xiang et al(1993)] Xiang Y, Poole D, Beddoes MP (1993) Multiply sectioned Bayesian networks and junction forests for large knowledge based systems. *Computational Intelligence* 9(2):171–220
- [Yannakakis(1981)] Yannakakis M (1981) Computing the minimum fill-in is NP-complete. *SIAM J of Algebraic and Discrete Methods* 2(1)
- [Zhang et al(2005)] Zhang W, Wang G, Xing Z, Wittenburg L (2005) Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161(1-2):55–87