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)], meeting scheduling [Wallace and Freuder(2005)], and university timetabling (Section 8). 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 is a case study on solving distributed university timetabling problems to illustrate the techniques presented so far. Section 9 addresses construction of agent organization for MSCNs. Section 10 shows how to convert any DisCSP into an MSCN. Proofs are collected in Appendix 1. Formal notations are listed in Appendix 2 for readers' convenience.

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 $\overline{x} \in D_X$ 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., $\cup_{R_X \in \Lambda} X = V$.

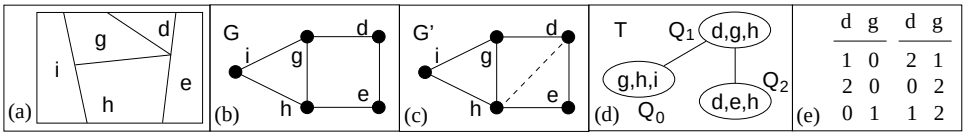

Fig. 1 (a) A map coloring CN; (b) primal graph; (c) triangulated graph; (d) resultant JT; (e) constraint $d \neq g$.

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* $\{red, green, blue\}$*, 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

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

Denote *projection* of config $\overline{x}$ to $Y \subseteq X$ by $\pi_Y(\overline{x})$. For instance, $\overline{x} = (d = 0, e = 2, g = 1)$ is a config of $X = \{d, e, g\}$. Its projection to $Y = \{e, g\}$ is the config $\overline{y} = (e = 2, g = 1)$. 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 $\overline{x} \in D_X$ *satisfies* constraint R_Y if either $X \cap Y = \emptyset$ (R_Y is irrelevant) or $\pi_{X \cap Y}(\overline{x}) \in \pi_{X \cap Y}(R_Y)$ (the projection of $\overline{x}$ to $X \cap Y$ matches the projection of one config in R_Y). A config $\overline{x}$ is *legal* if it satisfies every constraint in Λ. 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 $\mathcal{R} = (\mathcal{A}, V, \Omega, \Lambda, \Theta)$. $\mathcal{A} = \{A_0, ..., A_{\eta-1}\}$ is a set of $\eta > 1$ agents. The set V of env variables are decomposed into a collection of *subenvs*, $\Omega = \{V_0, ..., V_{\eta-1}\}$, such that $\cup_{i=0}^{\eta-1} V_i = V$. The set Λ of constraints are decomposed into $\Theta = \{\Lambda_0, ..., \Lambda_{\eta-1}\}$, where for each constraint R_X in Λ_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* $\mathcal{R}_i = (V_i, \Lambda_i)$. 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 $< x, y > \in E_i$ and $x, y \in V_j$, then $< x, y > \in E_j$. That is, constraints between shared variables are identical among agents involved. We refer to the primal graph depicting (V, Λ) 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 $V_0 = \{c, f, n, p\}$*

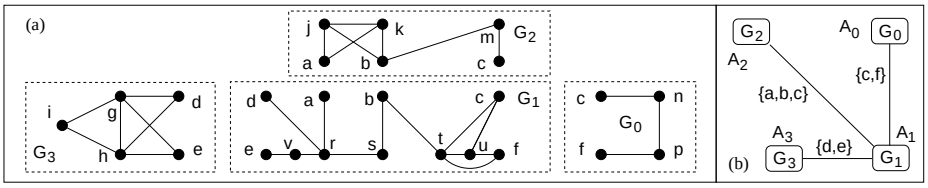

Fig. 2 (a) Local primal graphs of a DisCN that is an MSCN, where each link depicts a $\neq$ constraint. (b) The hypertree of MSCN.

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 Ω 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 Ω can be reordered as $V'_0, ..., V'_{\eta-1}$ 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 $< V_1, V_2, V_3 >$ 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 10, 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 9.2. 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 Λ, 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 Λ_Q;
4 replace Λ_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 $\overline{q}$ and that of R_C by $\overline{c}$;
*8 assign **project-equal** constraint $\pi_S(\overline{q}) = \pi_S(\overline{c})$ 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 Λ' of project-equal constraints specified over pairs of elements in $\mathcal{Q}$ define a *derived* binary CN $(\mathcal{Q}, \Lambda')$.

Example 3 *Consider the CN in Fig. 1. Its primal graph G in (b) is converted into JT T in (d). Applying Proc. 1 to the CN and T, its 6 binary constraints are assigned to clusters in T as follows:*

$$Q_0 : \{g \neq h, g \neq i, h \neq i\}; \quad Q_1 : \{d \neq g\}; \quad Q_2 : \{d \neq e, e \neq h\}.$$

The domain of relation variable v_{Q_2} is the relation R_{Q_2}:

$$R_{Q_2} \text{ over } (d, e, h) = \{(0,1,0), (0,1,2), (0,2,0), (0,2,1), (1,0,1), (1,0,2), (1,2,0), (1,2,1), (2,0,1), (2,0,2), (2,1,0), (2,1,2)\}.$$

The project-equal constraint between v_{Q_1} and v_{Q_2} requires that the config over Q_1 and that over Q_2 agree on values of d and h, e.g., $\overline{q_1}$ over $(d, g, h) = (0, 1, 1)$ and $\overline{q_2}$ over $(d, e, h) = (0, 2, 1)$.

Prop. 2 states that the solution set of $(\mathcal{Q}, \Lambda')$ is identical to that of $\mathcal{R}$. Its proof uses *idempotency* of natural join: Joining a relation multiple times has the effect of doing once.

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 $\overline{q}$ of Q and $\overline{c}$ of C are *consistent* if $\pi_S(\overline{q}) = \pi_S(\overline{c})$ (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.

Example 4 *Suppose T in Fig. 1 (d) is directed with Q_0 as the root. Then the parent of Q_1 is Q_0 and the parent of Q_2 is Q_1. T is locally directional arc-consistent relative to Q_0 if, for each config in R_{Q_1} there is a consistent config in R_{Q_1}, and for each config in R_{Q_1} there is a consistent config in R_{Q_2}. T is regionally directional arc-consistent relative to Q_0 if, in addition, for each config in R_{Q_0} there is a consistent config in R_{Q_2}.*

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 ∇ 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 $(\mathcal{Q}, \Lambda')$.

Proposition 4 (No Solution) *Let T be a JT representation of CN $\mathcal{R}$, $(\mathcal{Q}, \Lambda')$ 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 $O(t\ k^q)$.

Example 5 *Suppose Proc. 1 has been applied to the CN and its JT T in Fig. 1. Suppose Proc. 2 is then called in Q_0. Q_0 will call Proc. 2 on Q_1, which will in turn call Proc. 2 on Q_2.*

In response, Q_2 sends $\pi_{\{d,h\}}(R_{Q_2})$ to Q_1. Upon receiving, Q_1 modifies R_{Q_1} into $R_{Q_1} \bowtie \pi_{\{d,h\}}(R_{Q_2})$. It then sends $\pi_{\{g,h\}}(R_{Q_1})$ to Q_0. Upon receiving, Q_0 modifies R_{Q_0} into $R_{Q_0} \bowtie \pi_{\{g,h\}}(R_{Q_1})$. For this CN, Q_0 eventually returns ∇, and T is then locally directional arc-consistent. Since it is a JT, it is also regionally directional arc-consistent.

Suppose Proc. 3 is called next in Q_0. Q_0 removes all configs in R_{Q_0} except one. It projects the config onto $\{g, h\}$ and calls Proc. 3 in Q_1 with the projection. In response, Q_1 joins the projection with R_{Q_1}, and removes all configs in R_{Q_1} except one. It projects the config onto $\{d, h\}$ and calls Proc. 3 in Q_2 with the projection. When Q_2 receives the call, it operates similarly. Now the solution to the CN can be obtained by retrieving the single config in each cluster and joining them.

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 $\overline{s}$ such that no config $\overline{q}$ in R_Q satisfies $\pi_S(\overline{q}) = \overline{s}$. 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 6 *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 communicates 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 pairwise connected is identified, and these clusters are connected into a local JT T_i (bounded by box in Fig. 4).*

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

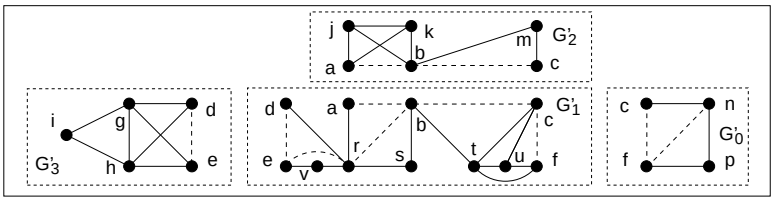

Fig. 3 Local primal graphs of MSCN are triangulated. Dashed links between nodes are fill-ins.

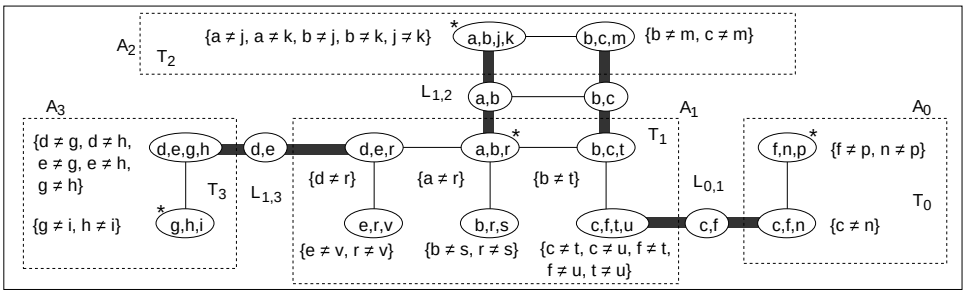

Fig. 4 LJF constructed from Fig. 2. Linkage hosts are indicated by thick lines. Constraints assigned to each cluster are in {}.

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 LJF 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 the structural conversion, constraints in each local CN are transferred to the local JT that is used for problem solving. Each agent A_i assigns constraints in Λ_i to clusters in T_i by AssignConsToJT (Section 4).

Example 7 *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
0	*0*	*1*	*2*
0	*0*	*2*	*1*
1	*1*	*0*	*2*
1	*1*	*2*	*0*
2	*2*	*0*	*1*
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 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			
d	e	g	h
a	b	j	k
c	f	t	u
0	0	1	2
0	0	2	1
1	1	0	2
1	1	2	0
2	2	0	1
2	2	1	0

R_2		
g	h	i
b	c	m
e	r	v
b	r	s
f	n	p
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_3		
d	e	r
a	b	r
b	c	t
c	f	n
0	0	1
0	0	2
0	1	1
0	1	2
0	2	1

0	2	2
1	0	0
1	0	2
1	1	0
1	1	2
1	2	0
1	2	2
2	0	0
2	0	1
2	1	0

2	1	1
2	2	0
2	2	1

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 $\{L_{i,j}\}$.
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 $\overline{q}$ and that of R_C by $\overline{c}$;
    assign constraint $\pi_S(\overline{q}) = \pi_S(\overline{c})$ 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.

Note that since T_j as well as cluster C are *private* to A_j, AssignConsToLT is a *logical* view of the underlying *physical* operation. T_j, C, R_C, $\overline{c}$ and v_C are virtual objects to A_i, not physical data, used to control message passing between A_i and A_j shown below.

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, $T = \{T_0, ..., T_{\eta-1}\}$ is a set of local JTs, one per subenv in Ω, 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. $\Delta = \{\Delta_0, ..., \Delta_{\eta-1}\}$ is a collection of constraint sets, one per T_i, expressing intra-agent constraints. Each Δ_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 Λ'_i, the set of project-equal constraints associated with $L_{i,j}$ as $\Lambda'_{i,j}$, and the union of these sets as $\Lambda' = (\cup_i \Lambda'_i) \cup (\cup_{i,j} \Lambda'_{i,j})$. Then $(\mathcal{Q}, \Lambda')$ defines a binary CN *derived* from LJF $\mathcal{F}$. Theorem 1 states that the solution set of $(\mathcal{Q}, \Lambda')$ is identical to that of $\mathcal{R}$.

Theorem 1 *Let $\mathcal{R}$ be an MSCN, $\mathcal{F}$ be its LJF, and $(\mathcal{Q}, \Lambda')$ be the binary CN derived from $\mathcal{F}$. Let Sol be the solution set of $\mathcal{R}$ and Sol' be the solution set of $(\mathcal{Q}, \Lambda')$. 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 Λ. Complexity of AssignConsToJT performed by all agents is $O(\eta\, \lambda\, k^q)$, where λ binds $|\Lambda_i|$, 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 $\overline{v_i}$ of V_i ($\overline{v_i} \in \bowtie_{R \in \Lambda_i} R$), 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 8 *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 $A_1, \ldots, A_m$, if any.

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

1 *for each agent A_i ($i = 1, \ldots, m$),*
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 Γ_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 Γ_c;
14 send Γ_c to A_c;
15 else return a special set ∇ to coordinator signifying successful completion;

Example 9 *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).

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

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 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 ∇.

Note that our emphasis is to present and illustrate the general algorithm, rather than covering all possible improvements for specific problem instances. For example, LT between A_1 and A_3 has a single linkage, a degenerated case. As a result, UnifyCons performed by A_3 can be simplified than illustrated above. Elaboration of such improvements is beyond scope of this paper.

Two important properties of CollectIntCons are established below: Lemma 1 says that relations at consistent clusters are equivalent when projected to their intersection. It is used in the proof of Lemma 2.

Lemma 1 (Projection) *Let Q and C be consistent clusters in a JT representation of a CN and $S = Q \cap C \neq \emptyset$. Then $\pi_S(R_Q) = \pi_S(R_C)$.*

Lemma 2 establishes properties of message Γ_c that A_0 sends to A_c during CollectIntCons. Property 1 says that Γ_c represents local CN constraints projected to the agent interface. Property 2 asserts consistency among elements of Γ_c.

Lemma 2 (Message) *Let T_0 be regionally fully arc-consistent. Then execution of lines 10 through 13 of CollectIntCons results in Γ_c such that*

1. $\bowtie_{R\in\Gamma_c} R = \pi_{I_c}(\bowtie_{R'\in\Lambda_0} R')$ *where* $I_c = V_0 \cap V_c$, *and*
2. *the linkage tree* $L_{0,c}$ *associated with* Γ_c *is regionally fully arc-consistent.*

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* ∇ *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 ∇ 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 $A_1, ..., A_m$, introduced in Section 6 will be used.

In response to message Γ 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 Γ 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 Γ;
4 else perform GetLocalSol with $\emptyset$;
5 for each agent A_i $(i = 1, \ldots, m)$,
6 initialize $\Gamma' = \emptyset$;
7 for each linkage S *with* A_i *(whose host cluster is* Q*), add* $\pi_S(R_Q)$ *to* Γ';
8 call DistribSol on A_i *with* Γ';

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 10 *To illustrate SolveDisCSP, suppose coordinator executes by choosing $A^* = A_0$. Example 9 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_9

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

R_{10}

a	b	r
b	c	m
b	c	t
c	f	n
d	e	r
2	2	1

R_{11}

b	r	s
e	r	v
f	n	p
2	1	0

R_{12}

a	b
b	c
c	f
d	e
2	2

R_{13}

a	b	j	k
c	f	t	u
d	e	g	h
2	2	1	0

R_{14}

g	h	i
1	0	2

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. Lemma 3 is used in its proof. Lemma 3 shows that arc-consistency in JT representations is preserved under natural join.

Lemma 3 *Let R_Q, R'_Q, R_C and R'_C be relations such that $Q' \subseteq Q$, $C' \subseteq C$, $Q \cap C = Q' \cap C' = S \neq \emptyset$, $\pi_S(R_Q) \subseteq \pi_S(R_C)$, and $\pi_S(R_{Q'}) \subseteq \pi_S(R_{C'})$. Then, we have*

$$\pi_S(R_Q \bowtie R_{Q'}) \subseteq \pi_S(R_C \bowtie R_{C'}).$$

In Lemma 3, Q' and C' can be viewed as a pair of adjacent clusters in a JT representation T with separator S. $\pi_S(R_{Q'}) \subseteq \pi_S(R_{C'})$ signifies that Q' is consistent relative to C'. Similarly, Q and C are a pair of adjacent linkages in a LT with separator S, and Q is consistent relative to C. Lemma 3 asserts that relative consistency between Q' and C' is preserved after natural join. This is used in the proof of Theorem 3.

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 $R' \subseteq Sol$, where Sol is the solution set of the MSCN.*

Let η 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 $O(\eta\ t\ k^{2q})$. This is summarized below.

Proposition 9 (Complexity of SolveDisCSP) *Let $\mathcal{F}$ be a LJF of an MSCN, η 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. The time complexity of SolveDisCSP is $O(\eta\ t\ k^{2q})$.*

Note that SolveDisCSP preserves agent privacy. For experimental implementation and empirical evaluation of SolveDisCSP, see [Mohamed(2011)].

8 Example: Distributed University Timetabling

Many applications of DisCSP exist, including sensor network coordination [Bejar et al(2005)], transportation vehicle scheduling [Calisti and Neagu(2004)], and meeting scheduling for participants [Wallace and Freuder(2005)]. In this section, we demonstrate the application of MSCN for solving a distributed university timetabling problem (DisUTTP), as specified below.

A university has a number of departments, and each offers a number of courses in a given semester. The semester is divided into several weeks. Courses are scheduled for one week and the timetable is repeated for each week. Each week is divided into a set of prefixed time slots of equal length. Each course consists of one or more lectures per week. Each lecture lasts for one time slot and occupies a lecture room for the slot. Scheduling is subject to four types of constraints.

1. (Room-slot) No two lectures can be offered in the same room at the same slot.
2. (Instructor) Lectures by the same instructor cannot be scheduled into the same slot.
3. (Course) Lectures of the same course cannot be offered at the same slot.
4. (Course group) If two courses are to be taken by students in the same semester according to program requirement, then their lectures cannot be scheduled into the same slots.

Traditionally, timetabling is solved centrally, where constraints are collected from departments and rooms are centrally managed. Collecting departmental constraints centrally is subject to cost of communication, time delay, and inflexibility.

To solve the problem as a DisUTTP, each department is allocated with a set of rooms, and timetables lectures of its courses. Due to constraints among courses managed by different

departments, timetables of departments involved must be coordinated. For instance, students in department $Dept_1$ are required to take courses Crs_1 and Crs_2 in a semester, while students in $Dept_2$ are required to take Crs_2 and Crs_3 in the same semester. Hence, Crs_2 must be scheduled identically at both departments, its lectures cannot be scheduled into the same slots with lectures of Crs_1 at $Dept_1$, and nor be scheduled into the same slots with lectures of Crs_3 at $Dept_2$. That is, the two departments must coordinate timetabling of Crs_1, Crs_2 and Crs_3.

Solving timetabling as DisUTTPs can avoid communicating local constraints centrally, shortens scheduling process, and improves flexibility. To do so, we first encode the DisUTTP into an MSCN. As the weekly timetable is repeated, scheduling can focus on lectures in a week over prefixed slots. Each lecture during the week is represented by a variable x. The collection of such variables form the set V of env variables.

Each x is associated with a tuple $(C_x, I_x, t_x, r_x, CO_x)$. Course ID C_x specifies the course that the lecture x belongs to. Instructor ID I_x specifies the instructor to teach the course. Element t_x is itself a variable, representing the slot that x will occur, and is associated with the domain D_{t_x}. Let ST denote the set of slots in a week. D_{t_x} is a heuristically determined, small subset of ST. Element r_x is also a variable, representing the room where x will be offered, and is associated with the domain D_{r_x}. Let RM denote the set of rooms allocated to the department that is in charge of scheduling x. D_{r_x} is a heuristically determined, small subset of RM. CO_x is a set of IDs for courses to be co-taken with C_x. From the associated tuple $(C_x, I_x, t_x, r_x, CO_x)$, the domain D_x of x is obtained as the Cartesian product $D_x = D_{t_x} \times D_{r_x}$, representing all possible ways in which the lecture may be timetabled.

Once variables in each subenv (one department) are determined, constraints between them are specified to form the local CN. This amounts to connecting each pair of constrained lecture variables x and y, elaborated below in relation to constraint types:

1. (Room-slot) If x and y satisfy $D_x \cap D_y \neq \emptyset$, their constraint is $(r_x \neq r_y) \vee (t_x \neq t_y)$.
2. (Instructor) If x and y satisfy $I_x = I_y$, their constraint is $t_x \neq t_y$.
3. (Course) If x and y satisfy $C_x = C_y$, their constraint is $t_x \neq t_y$.
4. (Course group) If x and y satisfy $C_x \in CO_y$, their constraint is $t_x \neq t_y$.

To provide readers with a more concrete idea, we describe the MSCN for a simulated DisUTTP that involves 8 departments. Each department offers between 8 and 13 courses for the semester, taught by between 5 and 9 instructors. Each department has between 1 and 3 courses to be taken by its own students. Each department also has between 1 and 3 courses to be taken by students in other departments. Each course has 2 or 3 lectures per week. This amounts to a total of 193 lectures, taught by a total of 53 instructors. As for the resources, 9 weekly time slots and 35 lecture rooms are available for lectures. Each department manages 3 to 5 rooms.

The MSCN for the DisUTTP consists of 8 agents each responsible for one department and associated with a local CN. Each local CN contains between 20 to 31 variables. Each variable represents a lecture, either offered by the department, or offered by another department but to be scheduled in coordination. The domain size for each lecture is $|D_x| = 6$. The local CN with the most constraints has 29 variables and 95 constraints. The local CN with the least constraints has 21 variables and 52 constraints. Fig. 5 shows the hypertree of the MSCN.

Each local JT of the LJF contains between 7 and 10 clusters. The largest cluster size in local JTs is 8, whose cluster space is 1.68×10^6. The MSCN is solved using a Dell Precision T7400 Workstation with 8 cores at 3.20 GHz. Each agent is run in one core. The total computation time to solve the MSCN is 615 seconds.

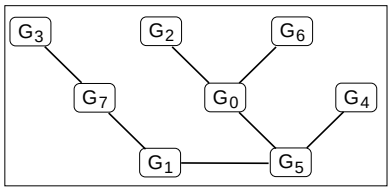

Fig. 5 The hypertree of MSCN for the example DisUTTP

9 Hypertree Agent Organization

9.1 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 $\mathcal{W} = \{W_i | i = 0, ..., \eta - 1\}$ as the *boundary collection* of the DisCN.

Algorithm 2 (HasHypertree)
for each agent A_i, W_i = 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 11 *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. 6 (a) and has a single cluster. HasHypertree returns has-hypertree.

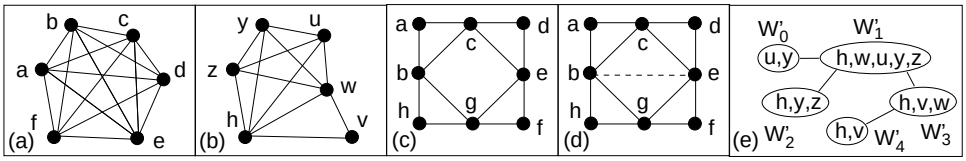

Fig. 6 Graph G_b for Examples 11 (a), 12 (b), and 14 (c). Triangulated graph (d) and JT (e) for Example 14.

Example 12 *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. 6 (b). It is triangulated and has two clusters. One of them,* $\{h, v, w\}$, *does not corresponds to any* W_i. *Hence, HasHypertree returns no-hypertree.*

Prop. 10 establishes soundness of HasHypertree.

Proposition 10 *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.

9.2 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 13 *Consider the DisCN in Example 11 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. 11 shows that after SetHypertree, a hypertree emerges for the DisCN.

Proposition 11 *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.*

Prop. 11 can be proven by noting the (distributed) correspondence of SetHypertree with the well-known maximum spanning tree algorithm for JT construction. When an agent becomes a winner, it is equivalent to adding its subenv to the current partial 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.

10 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 12). 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

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

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 14 illustrates the technical issue in doing so.

Example 14 (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. 6 (c). G_b is not triangulated and can be triangulated into G_b' in (d) by adding fill-in $< b, e >$. 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 12, 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 15 *Consider the DisCN in Example 12. Its G_b is Fig. 6 (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. 6 (e) can then be constructed.

Example 16 (More on EnlargeBoundary) *Consider Algorithm 4 applied to the DisCN in Example 14. G'_b is shown in Fig. 6 (d), the initial JT T' is shown in Fig. 7 (a), and initially,*

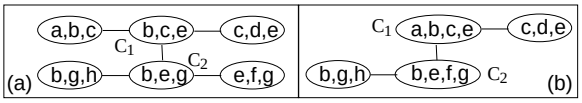

Fig. 7 (a) Initial JT T' for Example 15. (b)Updated JT T'.

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

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. 7 (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. 7 (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. 12 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 12 (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.

11 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 9 and 10) 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 η 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, extending Prop. 9, 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.

12 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 Intelligences 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

Appendix 1: Proofs

Proof of Proposition 1

1. First, we show that each element of Sol is a solution. Let $\overline{v}$ be a config in Sol. Then, for each constraint $R_X \in \Lambda$, we have $\pi_X(\overline{v}) \in R_X$. Hence, $\overline{v}$ satisfies every constraint.

Next, we show that every solution is in Sol. Suppose that there exists a solution $\overline{v'} \notin Sol$. Since for each constraint $R_X \in \Lambda$, $\pi_X(\overline{v'}) \in R_X$ holds, we have $\overline{v'} = \bowtie_{R_X \in \Lambda} \pi_X(\overline{v'})$. This implies $\overline{v'} \in Sol$: a contradiction.

2. The second statement follows directly from the first. □

Proof of Proposition 2

Consider adjacent clusters Q and C in T with separator S. Their relation variables in $(\mathcal{Q}, \Lambda')$ are v_Q and v_C with domains R_Q and R_C, respectively. Each partial solution $\overline{x}$ of $(\mathcal{Q}, \Lambda')$ over $\{v_Q, v_C\}$ satisfies $\pi_Q(\overline{x}) \in R_Q$, $\pi_C(\overline{x}) \in R_C$, and $\pi_S(\pi_Q(\overline{x})) = \pi_S(\pi_C(\overline{x}))$. Since each $\overline{x'} \in R_Q \bowtie R_C$ satisfies $\pi_Q(\overline{x}') \in R_Q$, $\pi_C(\overline{x}') \in R_C$, and $\pi_S(\pi_Q(\overline{x}')) = \pi_S(\pi_C(\overline{x}'))$, the project-equal constraint $\pi_S(\overline{q}) = \pi_S(\overline{c})$ is equivalent to $R_Q \bowtie R_C$. By substituting each project-equal constraint $\pi_S(\overline{q}) = \pi_S(\overline{c})$ in Λ' with $R_Q \bowtie R_C$, applying Prop. 1 to $(\mathcal{Q}, \Lambda')$, and discounting multiple occurrences of the same relation R_Q due to idempotency of natural join, we obtain the solution set of $(\mathcal{Q}, \Lambda')$ as $\bowtie_{Q \in T} R_Q$.

Since R_Q is obtained by joining constraints assigned to Q and each constraint in Λ is assigned to one cluster in T, we have $Sol' = \bowtie_{Q \in T} R_Q = \bowtie_{R \in \Lambda} R = Sol$.

□

Proof of Proposition 3

Let Q and C be clusters in T, where Q is an ancestor of C. We prove that Q is consistent relative to C by induction on the length l of path from Q to C. When $l = 1$, Q is the parent of C and Q is consistent relative to C by locally directional arc-consistency.

Assume that Q is consistent relative to C when $l \leq k$ and consider $l = k + 1$. Let the child cluster of Q on the path from Q to C be Q'. By locally directional arc-consistency, for each config $\overline{q}$ of Q, there exists a config $\overline{q}'$ of Q' such that $\pi_{Q \cap Q'}(\overline{q}) = \pi_{Q \cap Q'}(\overline{q}')$.

By inductive assumption, Q' is consistent relative to C. That is, for the above config $\overline{q}'$, there exists a config $\overline{c}$ of C such that $\pi_{Q' \cap C}(\overline{q}') = \pi_{Q' \cap C}(\overline{c})$. Since T is a JT (with running intersection property), we have $Q \cap C \subseteq Q'$. Hence, $\pi_{Q \cap C}(\overline{q}) = \pi_{Q \cap C}(\overline{q}') = \pi_{Q \cap C}(\overline{c})$. □

Proof of Proposition 4

[Sufficiency] Suppose $Sol = \emptyset$. This implies $\bowtie_{Q \in T} R_Q = \emptyset$. We prove by induction on the number of clusters in T. If T has a single cluster, then the above means $R_{Q^*} = \emptyset$. CollectSepCons called on Q^* will skip the *for* loop and return $\emptyset$ due to the *if* statement following the loop.

Assume that sufficiency holds when T has k clusters or less. Consider the case where T has $k+1$ clusters and Q^* has adjacent clusters $C_1, ..., C_j$. CollectSepCons called on Q^* is equivalent to performing $R_{Q^*} \bowtie R_{C_1^+} \bowtie ... \bowtie R_{C_j^+}$, where $R_{C_i^+}$ is the relation resultant from joining cluster relations of all clusters located at the subtree rooted at C_i. Without losing generality, assume that the above joins are performed from left to right. Since $\bowtie_{Q \in T} R_Q = \emptyset$, one of the following must be true:

1. None of $R_{C_1^+}, ..., R_{C_i^+}$ is empty, $R_{Q^*} \bowtie R_{C_1^+} \bowtie ... \bowtie R_{C_i^+}$ is empty, and $i \geq 1$ is the lowest such index.
2. None of $R_{C_1^+}, ..., R_{C_i^+}$ is empty, $R_{Q^*} \bowtie R_{C_1^+} \bowtie ... \bowtie R_{C_i^+}$ is nonempty, $R_{C_{i+1}^+}$ is empty, and $i+1 \geq 1$ is the lowest such index.

In the first case, the second if test in CollectSepCons succeeds. In the second case, the first if test succeeds by inductive assumption. As the result, Q^* returns $\emptyset$.

[Necessity] Suppose Q^* returns $\emptyset$. We use induction again. If T has a single cluster, necessity is trivially true. Assume that it holds when T has k clusters or less. We consider the case where T has $k+1$ clusters and Q^* has adjacent clusters $C_1, ..., C_j$. CollectSepCons called on Q^* is equivalent to performing $R_{Q^*} \bowtie R_{C_1^+} \bowtie ... \bowtie R_{C_j^+}$, where $R_{C_i^+}$ is defined as above. If Q^* returns $\emptyset$, it is due to a positive test either in the first if statement in CollectSepCons or in the second if statement. Each corresponds to one case enumerated in the proof of sufficiency, from which it follows $R_{Q^*} \bowtie R_{C_1^+} \bowtie ... \bowtie R_{C_j^+} = \emptyset$. That is, $Sol = \bowtie_{Q \in T} R_Q = \emptyset$. □

Proof of Proposition 5

It follows directly from how cluster Q updates its relation in CollectSepCons, namely, $R_Q = R_Q \bowtie R_S$. □

Proof of Proposition 6

The first statement follows from Prop. 1 and Prop. 4.

The second statement follows from Prop. 5, how cluster C updates its relation in DistribSepCons, namely, $R_C = R_C \bowtie R_S$, and Corollary 1. □

Proof of Proposition 7

By merging primal graphs of local CNs, a CN $\mathcal{R}'$ is well defined. Denote its solution set by Sol'. Each config in Sol' satisfies all constraints in all $\mathcal{R}_i$, and Sol' contains all such configs. From Prop. 1, difference between Sol' and Sol is that constraints over shared variables are joined multiple times in Sol, which is inconsequential by idempotency of natural join. □

Proof of Theorem 1

From $\mathcal{R} = (\mathcal{A}, V, \Omega, \Lambda, \Theta)$, $V = \bigcup_i V_i$, $\Lambda = \bigcup_i \Lambda_i$, and Prop. 7, we have

$$Sol = \bowtie_i (\bowtie_{R \in \Lambda_i} R). \tag{1}$$

The solution set of $(\mathcal{Q}, \Lambda')$ is

$$Sol' = \bowtie_{R \in \Lambda'} R = (\bowtie_i (\bowtie_{R \in \Lambda'_i} R)) \bowtie (\bowtie_{i,j} (\bowtie_{R' \in \Lambda'_{i,j}} R')), \tag{2}$$

where each iteration of $\bowtie_{i,j}$ is over a $L_{i,j}$.

We rewrite Eqn. (2) similarly to the proof of Prop. 2. Each constraint $R \in \Lambda'_i$ corresponds to a pair of adjacent clusters Q and C in local JT T_i, associated with relation variables v_Q and v_C, whose domains are R_Q and R_C, respectively. As argued in the proof of Prop. 2, the project-equal constraint R over v_Q and v_C is equivalent to constraint $R_Q \bowtie R_C$ over $Q \cup C$. By substitution of each $R \in \Lambda'_i$ with a corresponding constraint $R_Q \bowtie R_C$, from the specification of AssignConsToJT and idempotency of natural join, it follows that $(\bowtie_{R \in \Lambda'_i} R)$ in Eqn. (2) is equivalent to $(\bowtie_{R \in \Lambda_i} R)$ in Eqn. (1) for each i. This implies that $(\bowtie_i (\bowtie_{R \in \Lambda'_i} R))$ in Eqn. (2) is equivalent to Sol.

Next, we consider $(\bowtie_{R' \in \Lambda'_{i,j}} R')$ in Eqn. (2). Let S be a linkage in a LT, and X and Y be corresponding host clusters. The project-equal constraint R' between $v_X \in R_X$ and $v_Y \in R_Y$ is equivalent to constraint $R_X \bowtie R_Y$ over $X \cup Y$. Since both R_X and R_Y have occurred in $(\bowtie_i (\bowtie_{R \in \Lambda'_i} R))$ and natural join is idempotent, project-equal constraint over v_X and v_Y has no impact. Hence, $Sol' = (\bowtie_i (\bowtie_{R \in \Lambda'_i} R)) = Sol$. □

Proof of Lemma 1

Since Q is consistent relative to C, for each config in R_Q, there exists a consistent config in R_C. Hence, $\pi_S(R_Q) \subseteq \pi_S(R_C)$. Since C is consistent relative to Q, we also have $\pi_S(R_C) \subseteq \pi_S(R_Q)$. □

Proof of Lemma 2

Since T_0 is regionally full arc-consistent, R_S from line 12 is nonempty. As the result, the *for* loop (lines 11 through 13) produces a nonempty Γ_c with an element for each linkage in L_c. From Prop.s 1 and 2, we have $\bowtie_{R' \in \Lambda_0} R' = \bowtie_{Q \in T_0} R_Q$ and hence $\pi_{I_c}(\bowtie_{R' \in \Lambda_0} R') = \pi_{I_c}(\bowtie_{Q \in T_0} R_Q)$. To prove property 1, we only need to show $\pi_{I_c}(\bowtie_{Q \in T_0} R_Q) = \bowtie_{R \in \Gamma_c} R$. Denote $V_0 \setminus I_c$ by $\{v_1, ..., v_m\}$ and define $m-1$ supersets of I_c as

$$X_1 = I_c \cup \{v_1\}, \quad X_2 = I_c \cup \{v_1, v_2\}, ..., \quad X_{m-1} = I_c \cup \{v_1, ..., v_{m-1}\}.$$

We have

$$\pi_{I_c}(\bowtie_{Q \in T_0} R_Q) = \pi_{I_c}(\pi_{X_1}(\pi_{X_2}(...\pi_{X_{m-1}}(\bowtie_{Q \in T_0} R_Q)...))),$$

where the inner most projection removes v_m from the scope of the resultant relation, the second inner most projection removes v_{m-1}, and so on, until all variables in $V_0 \setminus I_c$ are removed. We parallel removal of v_i from the scope with its removal from T_0: If v_i is contained in a single cluster C in T_0, it is removed from C. If the removal renders C a subset of an adjacent cluster Q, merge C into Q, in which case $\pi_C(R_Q) = R_C$ due to Lemma 1 and disappearance of R_C has no impact to the result.

Due to LT construction [Xiang(2002)] and the above cluster merger, each v_i is guaranteed to be contained in a single cluster at the time it is removed. Hence, after the outer most projection is complete, clusters left in T_0 are precisely linkages in L_c and remaining relations are precisely those in Γ_c. Property 1 now follows.

Property 2 holds because T_0 is regionally fully arc-consistent, each separator of L_c is a separator of T_0, and each element of Γ_c is a projection of relation in the corresponding linkage host in T_0. □

Proof of Theorem 2

We prove by induction on the maximum path length l in hypertree from root agent A_0 to a leaf agent. If $l = 0$, A_0 is the only agent. When CollectIntCons is called on A_0, it goes

directly to UnifyCons. According to Prop. 6, A_0 returns $\emptyset$ iff $\mathcal{F}$ has no solution. Otherwise, ∇ is returned. Property 2 (regional full arc-consistency of T_0) follows from Prop. 6. Properties 1 and 3 are trivially true.

Next, assume that the theorem is true for $l \leq k$ and consider $l = k + 1$. There are three exhaustive and exclusive cases where $\mathcal{F}$ has no solution.

1. The sub-hypertree rooted at an adjacent agent A_i of A_0 has no solution.
2. Each sub-tree has a partial solution, but the partial solution for sub-hypertree rooted at A_i cannot be extended to a linkage host in A_0.
3. Each sub-tree has a partial solution that can be extended to each linkage host in A_0, but A_0 has no partial solution over V_0 or the solution cannot be extended relative to $\cup_i \Gamma_i$.

For each adjacent agent A_i $(i = 1, ..., m)$ of A_0, sub-hypertree rooted at A_i satisfies $l \leq k$. By assumption, A_0 returns $\emptyset$ through line 3 iff case 1 is true. A_0 returns $\emptyset$ through line 6 iff case 2 is true. By Prop. 6, A_0 returns $\emptyset$ through line 8 iff case 3 is true.

Therefore, A_0 returns ∇ iff $\mathcal{F}$ has solution. In that case, by Lemma 2, we have

$$R_i = \bowtie_{R \in \Gamma_i} R = \pi_{I_i}(\bowtie_{R' \in \Lambda_i} R'),$$

where $I_i = V_0 \cap V_i$ and Λ_i is the set of constraints associated with clusters in T_i. From Prop.s 1 and 2, it follows that R_i is the partial solution set of T_i over I_i.

On the other hand, T_0 is a JT representation. From Prop.s 1 and 2, it has the solution set $Sol_0 = \bowtie_{R \in \Lambda_0} R$ before CollectIntCons. After CollectIntCons, Sol_0 is restricted to $Sol_0' = Sol_0 \bowtie R_i$. From the property of natural join, we have

$$\pi_{I_i}(Sol_0 \bowtie R_i) \subseteq R_i,$$

which implies that A_0 is interface-consistent relative to A_i. From the inductive assumption, F is locally directional interface-consistent. From Prop. 8, property 1 follows.

Property 2 follows from line 7 and Prop. 6. Property 2 follows from Lemma 2.

□

Proof of Lemma 3

We have $\pi_S(R_Q \bowtie R_{Q'}) = \pi_S(R_Q) \bowtie \pi_S(R_{Q'})$ and $\pi_S(R_C \bowtie R_{C'}) = \pi_S(R_C) \bowtie \pi_S(R_{C'})$. From $\pi_S(R_Q) \subseteq \pi_S(R_C)$ and $\pi_S(R_{Q'}) \subseteq \pi_S(R_{C'})$, the result follows. □

Proof of Theorem 3

From Theorem 2, it follows that failure is returned iff $\mathcal{F}$ has no solution. Otherwise, DistribSol is run and we prove remainder of the theorem by induction on number η of agents. When $\eta = 1$, there is one agent A^*. Only line 4 is executed. From Theorem 1, the theorem is trivially true.

Next, we assume that the theorem is true for $\eta \leq k$ and consider $\eta = k + 1$. Let $k + 1$'th agent A_{k+1} be a leaf agent in hypertree (directed from root A^*) and its adjacent agent be A_k. Denote part of LJF without A_{k+1} by $\mathcal{F}_k$ and its solution set by Sol_k. By inductive assumption, before A_k calls DistribSol in A_{k+1} (line 8), we have $R_k = \bowtie_{i=1}^{k} (\bowtie_{Q \in T_i} R_Q) \subseteq Sol_k$. Furthermore, because linkage tree $L_{k,k+1}$ between A_k and A_{k+1} is a JT, from Prop.s 1 and 2, Γ' (see DistribSol) represents a partial solution over $I_{k+1} = V_k \cap V_{k+1}$. In other words, Γ' represents a partial solution $\pi_{I_{k+1}}(R_k)$.

By Theorem 2 (1), A_k is interface-consistent with A_{k+1}. Hence, a partial solution in A_{k+1} exists that extends $\pi_{I_{k+1}}(R_k)$ to V_{k+1}. By Theorem 2 (3), $L_{k,k+1}$ (associated with Γ') is regionally fully arc-consistent. By Theorem 2 (2), T_{k+1} is regionally fully arc-consistent. Denote by Q^* the linkage host on which DistribSepCons is called in the last line of GetLocalSol. Direct T_{k+1} with Q^* as root. By application of Lemma 3 to each pair of linkage hosts in T_{k+1} downstream from Q^*, we conclude that when A_{k+1} executes line 3 of DistribSol and performs GetLocalSol, but before it calls DistribSepCons (the last line of GetLocalSol), T_{k+1} is regionally directional arc-consistent relative to Q^*. Hence, DistribSepCons will produce the partial solution that extends $\pi_{I_{k+1}}(R_k)$ to V_{k+1}. This effectively extends R_k into a solution to $\mathcal{F}$. By Theorem 1, it is a solution to the MSCN. □

Proof of Proposition 10

[Sufficiency] Suppose has-hypertree is returned. Then G_b is triangulated and hence there exists a JT T made of clusters from G_b. For each W_i, if it is not a cluster in T, then it must be a subset of a cluster C in T. Add cluster W_i to T and make it adjacent to C. Now each cluster of T corresponds to a unique W_i. Add to W_i private variables of A_i. The resultant is a JT which is a hypertree of the DisCN.

[Necessity] Suppose no-hypertree is returned. Then either G_b is not triangulated, in which case there exists no JT made of clusters from G_b, or it is triangulated but a cluster C does not corresponds to any W_i, e.g., Example 12. In the latter case, a JT exists made of clusters from G_b, but these clusters do not correspond to agents' subenv. □

Proof of Proposition 12

1. $\mathcal{W}'$ is initialized to $\mathcal{W}$ and is only modified in the second *for* loop. Each W_i' removed from $\mathcal{W}'$ is replaced by a cluster C. Hence, the condition holds.
2. Each W_i' is either the original W_i, or is created in the first *for* loop, which ensures $W_i' \supset W_i$.
3. After the second *for* loop, each cluster in T' is an element of $\mathcal{W}'$. For each $W_i' \in \mathcal{W}'$, if W_i' is not a cluster of T', it must be a proper subset of a cluster C in T'. By adding W_i' as an adjacent cluster of C in T', T' remains a JT. A JT can thus be created where each cluster corresponds to a $W_i' \in \mathcal{W}'$. □

Appendix 2: Notation and Abbreviation (roughly in order of appearance)

CSP :	Constraint satisfaction problem
$DisCSP$:	Distributed constraint satisfaction problem
$DisCOP$:	Distributed constraint optimization problem
V, env :	Set of environment variables
Λ :	Set of constraints
CN :	Constraint network
$DisCN$:	Distributed constraint network
$MSCN$:	Multiply sectioned constraint network
$\mathcal{R}$:	CN, DisCN, or MSCN
D_v :	Domain of variable v
D_X :	Space of variable set $X \subseteq V$
R_X :	Constraint with scope $X \subset V$
U_X :	Dumb constraint over X (a universal relation)
$\pi_Y(R_X)$:	Projection of R_X to $Y \subseteq X$
JT :	Junction tree
A_i :	An agent
$\mathcal{A}$:	Set of agents in a DisCN
η :	Number of agents in $\mathcal{A}$
V_i :	A subenv (sub-environment)
Ω :	Collection of subenvs
Λ_i :	Set of constraints in a subenv
Θ :	Collection of constraint sets one per subenv
$\mathcal{R}_i$:	Local CN of A_i
G_i :	Local primal graph of A_i
I_{ij} :	Border between A_i and A_j
$PVSC$:	Private variables, shared constraints
$\bowtie$:	Relational operator *natural join*
∇ :	Special set signifying successful operation
Sol :	Solution set of CN or MSCN
T_i :	Local junction tree of A_i
$\mathcal{F}, LJF$:	Linked junction forest
LT :	Linkage tree
W_i :	Boundary (shared variables) of A_i relative to other agents
G_b, G_b' :	Boundary graph
Co :	Coordinator agent
Itr :	Integrator agent