

Solving Multiagent Constraint Optimization Problems on the Constraint Composite Graph [★]

Ferdinando Fioretto¹, Hong Xu², Sven Koenig², and T. K. Satish Kumar²

¹ Georgia Institute of Technology
fioretto@gatech.edu

² University of Southern California
hongx@usc.edu, skoenig@usc.edu, tkskwork@gmail.com

Abstract. We introduce the *Constraint Composite Graph* (CCG) for *Distributed Constraint Optimization Problems* (DCOPs), a popular paradigm used for the description and resolution of cooperative multi-agent problems. The CCG is a novel graphical representation of DCOPs on which agents can coordinate their assignments to solve the distributed problem suboptimally. By leveraging this representation, agents are able to reduce the size of the problem. We propose a novel variant of Max-Sum—a popular DCOP incomplete algorithm—called *CCG-Max-Sum*, which is applied to CCGs, and demonstrate its efficiency and effectiveness on DCOP benchmarks based on several network topologies.

1 Introduction

In a cooperative *multiagent system*, multiple autonomous agents interact to pursue personal goals and to achieve shared objectives. The *Distributed Constraint Optimization Problem* (DCOP) model [17, 6] is an elegant formalism to describe cooperative multiagent problems that are distributed in nature. In this model, a collection of agents coordinate a value assignment to the problem variables with the goal of optimizing a global objective within the confines of localized communication. DCOPs have been used to solve a variety of problems in the context of coordination and resource allocation [30, 9], sensor networks [5], and device coordination in smart homes [22, 8].

DCOP algorithms are either *complete* or *incomplete*. Complete algorithms find an optimal solution to the problem employing one of two broad modus operandi: distributed search-based techniques [17, 26] or distributed inference-based techniques [20, 24]. In search-based techniques, agents traverse the search space by selecting value assignments and communicating them to other agents. Inference-based techniques rely instead on the notion of agent belief, describing

[★] The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, and 1817189. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

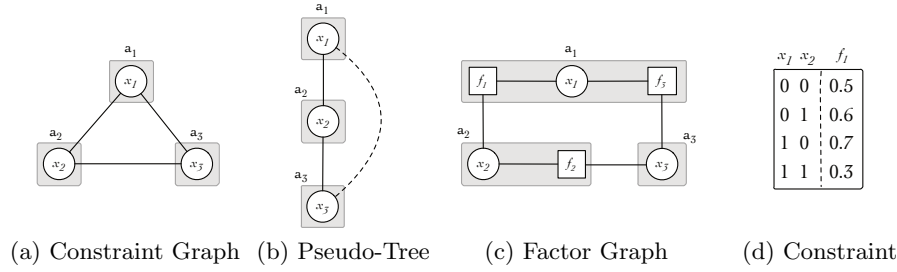


Fig. 1: DCOP constraint graph (a), pseudotree (b), factor graph (c), and a constraint (d).

the best cost an agent can achieve for each value assignment to its variables. These beliefs drive the value-selection process of the agents to find an optimal solution to the problem. Since finding an optimal DCOP solution is NP-hard [17], optimally solving a DCOP requires exponential time or space in the worst case. Thus, there is growing interest in the development of incomplete algorithms, which trade off solution quality for better runtimes. Similar to complete algorithms, incomplete algorithms can be classified as local search-based [16, 27] and inference-based [5]. Some incomplete algorithms have been used in different multiagent applications. For instance, Max-Sum [5] is an inference-based incomplete algorithm which has been successfully used to solve sensor networks problems [5] and smart home coordination problems [22].

In both complete and incomplete DCOP algorithms, the problem resolution process is characterized by the graphical representation of the problem. The three most important problem representations are the *constraint graph*, the *pseudo-tree*, and the *factor graph*. The first one represents a problem as a graph whose nodes describe the variables and whose edges describe the constraints. The second one is a rearrangement of the constraint graph, where a subset of edges forms a rooted tree and where two variables participating in the same constraint appear in the same branch of the tree. The third one represents the problem as a bipartite graph where nodes represent both variables and constraints, and edges link the constraint nodes to the variables participating in the associated constraint. In many local search algorithms, such as MGM [16], DSA [27], or the region-optimal algorithm family [19], agents operate directly on the constraint graph and perform distributed local searches by exchanging information with their neighbors in the constraint graph. In the main inference-based algorithms, the agents operate on either a pseudo-tree (e.g., P-DCOP [21]) or a factor graph (e.g., Max-Sum). In the former case, agents exchange messages following the structure of the pseudo-tree, typically alternating between a phase in which messages are propagated up from the leaf agents to the root agent of the pseudo-tree, and one in which information is propagated down. In the latter case, there are two types of entities, which represent variables and constraints. Both of them participate in the message-exchange process to solve the problem.

All these representations allow agents to exploit the graphical structure of the problem. However, they hide the numerical structure of the problem’s constraints. Thus, in this paper, we introduce the *Constraint Composite Graph (CCG)* for DCOPs, a lifted graphical representation that provides a framework for exploiting simultaneously the graphical structure of the agent-coordination process as well as the numerical structure of the constraints involving the variables controlled by the agents. CCGs have recently been introduced in the context of *Weighted Constraint Satisfaction Problems (WCSPs)* [13, 15], and shown to be highly effective in solving a wide range of problems [25]. We contribute to the development of inference-based DCOP algorithms by presenting a novel framework for solving DCOPs sub-optimally whose agent interactions are driven by the structure of the CCG representation. We analyze the behavior of our framework on federated social network problems (introduced in Section 5) and random Boolean problems on different graph topologies and show its effectiveness on several important classes of graphs, including grid networks and scale-free networks, which are used to model many applications in distributed settings.

To the best of our knowledge, this work describes the first proposal of a distributed message-passing algorithm based on the CCG representation. We refer to our algorithm as a “lifted” message passing algorithm since it works on the CCG representation of a DCOP.

2 Background

We now review the distributed constraint optimization framework, the graphical models commonly adopted to represent a DCOP, and the CCG model.

Distributed Constraint Optimization A *Distributed Constraint Optimization Problem (DCOP)* is a tuple $P = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{A}, \alpha \rangle$, where: $\mathbf{X} = \{x_1, \dots, x_n\}$ is a set of *variables*; $\mathbf{D} = \{D_{x_1}, \dots, D_{x_n}\}$ is a set of finite *domains* for the variables in $\mathbf{X}$; $\mathbf{F} = \{f_1, \dots, f_e\}$ is a set of *constraints* (also called *cost functions*), where $f : \prod_{x \in \mathbf{x}^f} D_x \rightarrow \mathbb{R}_+ \cup \{\infty\}$ and $\mathbf{x}^f \subseteq \mathbf{X}$ is the set of the variables (also called the *scope*) of f ; $\mathbf{A} = \{a_1, \dots, a_p\}$ is a set of *agents*; and $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a function that maps each variable to one agent. Fig. 1(d) shows an example constraint. It specifies the costs of all combinations of values for the variables x_1, x_2 in its scope. For a variable $x \in \mathbf{X}$, we use $\mathbf{f}^x$ to denote the set of constraints that involve x in their scopes.

A *partial assignment* σ_X is an assignment of values to a set of variables $X \subseteq \mathbf{X}$ that is consistent with their domains; i.e., it is a partial function $\theta : \mathbf{X} \rightarrow \cup_{i=1}^n D_{x_i}$ such that, for each $x_j \in \mathbf{X}$, if $\theta(x_j)$ is defined (i.e., $x_j \in X$), then $\theta(x_j) \in D_{x_j}$. For a set of variables $V = \{x_{i_1}, \dots, x_{i_h}\} \subseteq X$, $\pi_V(\sigma_X) = \langle \theta(x_{i_1}), \dots, \theta(x_{i_h}) \rangle$ is the *projection* of σ_X onto the variables in V , where $i_1 < \dots < i_h$. When $V = \{x_i\}$ is a singleton, we write $\pi_{x_i}(\sigma_X)$ to denote the projection of σ_X onto x_i . The *cost* $\mathcal{F}(\sigma_X) = \sum_{f \in \mathbf{F}: \mathbf{x}^f \subseteq X} f(\pi_{\mathbf{x}^f}(\sigma_X))$ of an assignment σ_X is the sum of the evaluation of the constraints involving all variables in X . A *solution* is a partial assignment σ_X (written σ for shorthand) for all

variables of the problem, i.e., with $X = \mathbf{X}$, whose cost is finite (i.e., $\mathcal{F}(\sigma) \neq \infty$). The goal is to find an optimal solution $\sigma^* = \operatorname{argmin}_{\sigma} \mathcal{F}(\sigma)$. In this paper, we restrict our attention to Boolean DCOPs (i.e., DCOPs where all domains are $\{0, 1\}$). Despite our focus on Boolean DCOPs, the concepts introduced in the next sections are generalizable, as discussed in Section 6.

Given a DCOP P , its *constraint graph* is $G_P = (\mathbf{X}, E_C)$, where an undirected edge $\{x, y\} \in E_C$ exists if and only if there exists an $f \in \mathbf{F}$ such that $\{x, y\} \subseteq \mathbf{x}^f$. The constraint graph provides a standard representation of a DCOP instance. It highlights the locality of interactions among agents and therefore is commonly adopted by DCOP algorithms. Fig. 1(a) shows an example constraint graph of a DCOP instance with three agents a_1, a_2 , and a_3 , each controlling one variable with domain $\{0, 1\}$. There are three constraints: f_1 with scope $\mathbf{x}^{f_1} = \{x_1, x_2\}$, f_2 with scope $\mathbf{x}^{f_2} = \{x_2, x_3\}$, and f_3 with scope $\mathbf{x}^{f_3} = \{x_1, x_3\}$.

A *pseudo-tree* for P is a spanning tree $T_P = (\mathbf{X}, E_T)$ of G_P , i.e., a connected subgraph of G_P that contains all nodes and is a rooted tree, with the following additional condition: for each $x, y \in \mathbf{X}$, if $\{x, y\} \subseteq \mathbf{x}^f$ for some $f \in \mathbf{F}$, then x and y appear in the same branch of T_P (i.e., x is an ancestor of y in T_P or vice versa). Figure 1(b) shows one possible pseudo-tree for our example DCOP, where the solid lines represent tree edges and the dotted line represents a *backedge* that connects an agent with one of its ancestors.

A factor graph [12] is a bipartite graph used to represent the factorization of a function. Given a DCOP instance P , the corresponding factor graph $F_P = (\mathbf{X}, \mathbf{F}, E_F)$ is composed of variable nodes $x \in \mathbf{X}$, function nodes $f \in \mathbf{F}$, and edges E_F such that there is an undirected edge between function node f and variable node x if and only if $x \in \mathbf{x}^f$. Fig. 1(c) illustrates the factor graph of our example DCOP instance, where each agent a_i controls its variable x_i and, in addition, a_1 controls the constraints f_1 and f_3 , and a_2 controls the constraint f_2 .

Max-Sum Max-Sum [5] is a popular incomplete DCOP algorithm. Its agents operate on a factor graph F_P through a synchronous iterative process. Albeit the logic of each variable node and each function node is executed within an agent, to ease exposition, in what follows, we treat them as entities that are able to send and receive messages. In each iteration, each function node f exchanges messages with the nodes of variables in its scope $\mathbf{x}^f$, and each variable node x exchanges messages with the nodes of constraints which involve x in their scopes $\mathbf{f}^x$. Thus, each node exchanges messages with its neighbors in the factor graph.

The content of the messages sent by each function (variable) node is based exclusively on the information received from neighboring variable (function) nodes. The message $q_{x \rightarrow f}^i$ sent by a variable node x to a function node f in $\mathbf{f}^x$ in iteration i contains, for each value $d \in D_x$, the aggregated costs for d received from all neighboring function nodes in iteration $i - 1$, excluding f . It is defined as a function $q_{x \rightarrow f}^i : D_x \rightarrow \mathbb{R}_+ \cup \{\infty\}$, whose value is 0 for all $d \in D_x$ when $i = 0$ and

$$q_{x \rightarrow f}^i(d) = \alpha_{xf}^i + \sum_{f' \in \mathbf{f}^x \setminus \{f\}} r_{f' \rightarrow x}^{i-1}(d) \quad (1)$$

when $i > 0$. Here, $r_{f' \rightarrow x}^{i-1}$ is the message received by variable node x from function node f' in iteration $i-1$ and α_{xf}^i is a *normalizing* constant used to prevent the values of the transmitted messages from growing arbitrarily and chosen so that $\sum_{d \in D_x} q_{x \rightarrow f}^i(d) = 0$ holds. The message $r_{f \rightarrow x}^i$ sent by a function node f to a variable node $x \in \mathbf{x}^f$ in iteration i contains, for each $d \in D_x$, the minimum cost of any assignments of values to the variables in $\mathbf{x}^f$ in which x takes value d . It is defined as a function $r_{f \rightarrow x}^i : D_x \rightarrow \mathbb{R}_+ \cup \{\infty\}$, whose value is 0 when $i=0$ and

$$r_{f \rightarrow x}^i(d) = \min_{\sigma_{\mathbf{x}^f} : \pi_x(\sigma_{\mathbf{x}^f}) = d} f(\sigma_{\mathbf{x}^f}) + \sum_{x' \in \mathbf{x}^f \setminus \{x\}} q_{x' \rightarrow f}^i(\pi_{x'}(\sigma_{\mathbf{x}^f})) \quad (2)$$

when $i > 0$. Here, $\sigma_{\mathbf{x}^f}$ is a possible assignment of values to all variables in the scope $\mathbf{x}^f$ of the constraint f , given that variable $x \in \mathbf{x}^f$ takes value d . The agent controlling a variable node x decides its value assignment at the end of each iteration $i > 0$ by computing its associated belief $b_x^i(d)$ for each $d \in D_x$: $b_x^i(d) = \sum_{f \in \mathbf{f}^x} r_{f \rightarrow x}^{i-1}(d)$ and choosing the assignment d^{*i} such that, $d^{*i} = \operatorname{argmin}_{d \in D_x} b_x^i(d)$. This form of message passing allows for an inference-based method: Max-Sum agents initialize all their messages to 0 and, in each iteration $i > 0$, retain only the most recent messages, overwriting the messages received in previous iterations.

Max-Sum is an incomplete DCOP algorithm. However, on acyclic problems, it is guaranteed to converge to an optimal solution [5].

3 The Constraint Composite Graph

We now describe the *constraint composite graph (CCG)*, a graphical structure that can be used to represent DCOPs. Its goal is to exploit simultaneously the graphical structure of the agent interactions as well as the numerical structure of the cost functions. It is a node-weighted tripartite graph $G_{\text{CCG}} = \langle V = \mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}, E, w \rangle$, where $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are a partition of V . The nodes in $\mathbf{X}$ correspond to the DCOP variables, while the nodes in $\mathbf{Y}$ and $\mathbf{Z}$ correspond to auxiliary variables introduced to model a reformulation of the original problem into a *Minimum Weighted Vertex Cover (MWVC)*.

The concept of a CCG was first proposed by Kumar [13] as a combinatorial structure associated with a *Weighted Constraint Satisfaction Problem (WCSP)*. WCSPs are similar to DCOPs, except that all computations are centralized. In that proposal, it was shown that the task of solving a WCSP can be reformulated as the task of finding a MWVC on its associated CCG [13–15]. A desirable property of the CCG is that it can be constructed in polynomial time and is always tripartite [13–15]. CCGs also enable the use of *kernelization methods* for solving WCSPs [25], which are polynomial-time procedures that can simplify a problem to a smaller one, called the kernel. The *Nemhauser-Trotter reduction (NT reduction)* [18, 2] is one such kernelization method and uses a maxflow procedure to find the kernel.

In the next section, we introduce an extension of the Max-Sum algorithm, called CCG-Max-Sum, which can be used directly on CCGs.

Algorithm 1: CCG-MAX-SUM

```

// CCG Construction Phase
1 foreach  $f_i \in \mathbf{F}_i$  do
2    $p_i \leftarrow \text{construct-polynomial}(f_i)$ ;
3    $G_{CCG_i} = \langle V_i = X_i \cup Y_i \cup Z_i, E_i, w_i \rangle \leftarrow \text{decompose-polynomial}(p_i)$  ;
4 foreach  $f \in \mathbf{F}_{CCG_i}$  involving variable  $v_j$  with  $\alpha(v_j) \neq a_i$  do
5    $a_i$  sends  $f$  to  $a_{\alpha(v_j)}$ ;
6 When agent  $a_i$  receives  $f$  involving  $v_i \in \mathbf{X}_i$  from neighboring agent  $a_j$ :
    $f_{v_i}(1) \leftarrow f_{v_i}(1) + f(1)$  ;
// Message Passing Phase
7  $\mu_{v_i \rightarrow v_j} \leftarrow 0$  ( $\forall v_i \in V_i, \forall v_j \in N(v_i)$ );
8 while termination condition is not met do
9   Wait for all messages  $\mu_{v_j \rightarrow v_i}$  from  $v_j \in N(v_i)$  ( $\forall v_i \in V_i$ );
10  foreach  $v_i \in V_i$  do
11    Update  $\mu_{v_i \rightarrow v_j}$  according to Eq. (5);
12 for  $v_i \in \mathbf{X}_i$  do
13   if  $w_{v_i} < \sum_{v_j \in N(v_i)} \mu_{v_j \rightarrow v_i}$  then  $v_i \leftarrow 1$  else  $v_i \leftarrow 0$ ;

```

4 CCG-Max-Sum

CCG-Max-Sum is an incomplete, iterative DCOP algorithm which works in two phases, namely, the *CCG construction* and the *message passing*, which are executed sequentially and summarized in Algorithm 1. In the CCG construction phase, the agents coordinate in the construction of a CCG and take ownership of the auxiliary variables and constraints introduced by this lifted graphical representation. Afterwards, in the message passing phase, the agents execute the iterative synchronous process which extends the Max-Sum algorithm.

In what follows, we use $G_i = \langle \mathbf{X}_i, \mathbf{F}_i \rangle$ to denote the subgraph of the constraint graph controlled by agent a_i , where the sets $\mathbf{X}_i \subseteq \mathbf{X}$ form a partition of the set of variables $\mathbf{X}$, and the sets $\mathbf{F}_i \subseteq \mathbf{F}$ form a partition for the constraint set $\mathbf{F}$.

4.1 CCG Construction Phase

The CCG construction proceeds in 3 stages:

1. **Expressing Constraints as Polynomials** In this stage, each agent a_i transforms the constraints $f_i \in \mathbf{F}_i$ it controls into polynomials p_i (line 2 of Algorithm 1) using standard Gaussian Elimination. We use $G_{CCG_i} = \langle V_i = X_i \cup Y_i \cup Z_i, E_i, w_i \rangle$ to denote the portion of the CCG obtained from constraint f_i . Consider the example constraint f_1 in Fig. 1(d), which involves the variables x_1 and x_2 . It can be written as a polynomial $p_1(x_1, x_2)$ in x_1 and x_2 of degree 1 each: $p_1(x_1, x_2) = c_{00} + c_{01}x_1 + c_{10}x_2 + c_{11}x_1x_2$. The coefficients c_{00} , c_{01} , c_{10} , and c_{11} of the polynomial can be computed by solving a system of linear equations, where each equation corresponds to an entry in the constraint table, using standard Gaussian Elimination. In our example:

$$p_1(0, 0) = 0.5 \quad p_1(0, 1) = 0.6 \quad p_1(1, 0) = 0.7 \quad p_1(1, 1) = 0.3$$

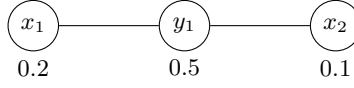


Fig. 2: The projection of an MWVC on the IS $\{x_1, x_2\}$ of this node-weighted undirected graph leads to Fig. 1(d). The weights on x_1 , x_2 , and y_1 are 0.2, 0.1, and 0.5, respectively. The entry 0.6 in cell $(x_1 = 0, x_2 = 1)$ in Fig. 1(d), for example, indicates that, when x_1 is necessarily excluded from the MWVC but x_2 is necessarily included in it, then the weight of the MWVC $\{x_2, y_1\}$ is 0.6.

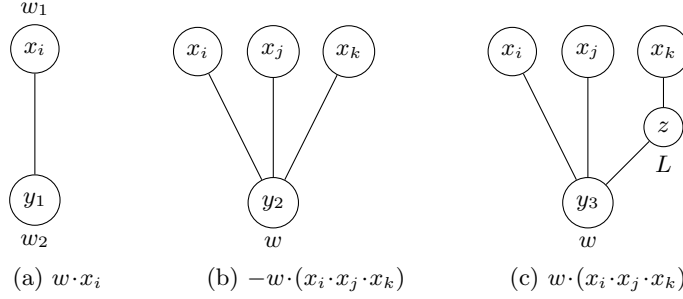


Fig. 3: The lifted graphical representation of terms in a polynomial for linear (a), negative nonlinear (b), and positive nonlinear (c) terms. We assume that $w > 0$ in (b) and (c) (but not in (a)). A node has a zero weight if no weight is shown. In (a), w_1 and w_2 satisfy $w_1 - w_2 = w$.

$$c_{00} = 0.5 \quad c_{01} = 0.1 \quad c_{10} = 0.2 \quad c_{11} = -0.5.$$

2. **Decomposing the Terms of the Polynomials** In this stage, for each $f_i \in \mathbf{F}_i$, the agent that controls it constructs a subgraph G_{CCG_i} of the CCG (line 3 of Algorithm 1). At the end of this stage, each agent introduces new sets of auxiliary variables Y_i and Z_i and replaces its constraints with a new set $\mathbf{F}_{\text{CCG}_i}$ of constraints that involve the decision variables and its newly introduced auxiliary variables. Before describing this procedure, we review the concept of the MWVC, a cornerstone concept for the notion of the CCG.

A *minimum node cover* of $G = \langle V, E \rangle$ is the smallest set of nodes $S \subseteq V$ such that every edge in E has at least one of its nodes in S . When G is node-weighted, (i.e., each node $v_i \in V$ has a non-negative weight w_i associated with it), its MWVC is defined as a node cover of minimum total weight of its nodes.

For a given graph G , one can project MWVCs on a given independent set (IS) $U \subseteq V$. (An IS is a set of nodes in which no two nodes are connected by an edge.) The input to such a projection is the graph G as well as an IS $U = \{u_1, u_2, \dots, u_k\}$ on G . The output is a table of 2^k numbers. Each entry in this table corresponds to a k -bit vector. We say that a k -bit vector t imposes the following restrictions: **(a)** If the i^{th} bit t_i is 0, then node u_i has to be excluded from the MWVC; and **(b)** if the i^{th} bit t_i is 1, then the node u_i has to be included in the MWVC. The projection of an MWVC on the IS U is then defined to be a table with entries corresponding to each of the 2^k possible k -bit

vectors $t^{(1)}, t^{(2)}, \dots, t^{(2^k)}$. The value of the entry that corresponds to $t^{(j)}$ is the weight of the MWVC conditioned on the restrictions imposed by $t^{(j)}$.

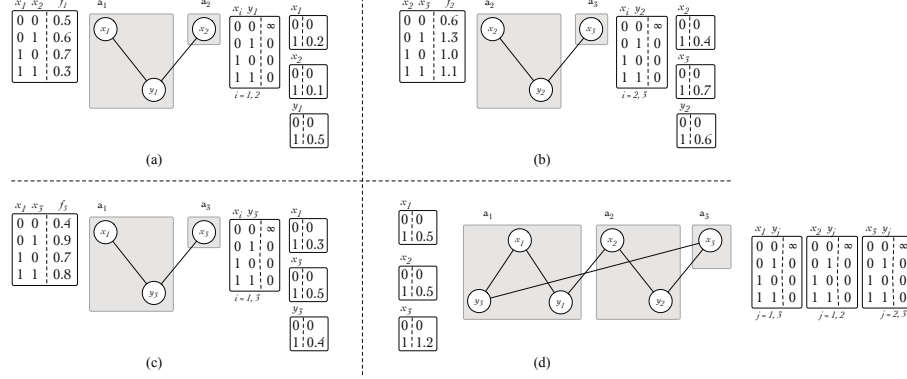


Fig. 4: (a)–(c): CCG gadget graph construction in the “Decomposing the Terms of Polynomials” stage for the example DCOP of Fig. 1. The original constraint, the associated CCG gadget, and the new constraint are shown on the left, middle, and right of each panel, respectively. (d): CCG construction in the “Merging Gadget Graphs into a CCG” stage for the example DCOP of Fig. 1. It is obtained by merging the CCG gadget graphs in (a)–(c).

Figure 2 illustrates this projection for the subgraph of our example DCOP problem of Fig. 1(a) that involves variables x_1 and x_2 and constraint f_1 , whose costs are shown in Fig. 1(d).

The table produced by projecting an MWVC on the IS U can be viewed as a constraint over $|U|$ Boolean variables. Conversely, given a (Boolean) constraint, we can design a lifted representation for it so as to be able to view it as the projection of an MWVC on an IS for some intelligently constructed node-weighted undirected graph [13, 14]. The lifted graphical representation of a constraint depends on the nature of the terms in the polynomial that describes the constraint. We distinguish three classes of terms: *linear terms*, *negative nonlinear terms*, and *positive nonlinear terms*. We can construct a lifted graphical representation, i.e., a *gadget graph*, for each term in the polynomial of each constraint as follows.

- A **linear term** is represented with the two-node graph shown in Fig. 3(a) by connecting the variable node with an auxiliary node.
- A **negative nonlinear term** is represented with the “flower” structure as depicted in Fig. 3(b). Consider the term $-w \cdot (x_i \cdot x_j \cdot x_k)$ where $w > 0$. Projecting an MWVC on the “flower” structure on the variable nodes represents $w - w \cdot (x_i \cdot x_j \cdot x_k)$. The constant term w does not affect the optimality of the solution.
- A **positive nonlinear term** is represented using the “flower+thorn” structure as depicted in Fig. 3(c). Consider the term $w \cdot (x_i \cdot x_j \cdot x_k)$ where $w > 0$. The projection of an MWVC on the “flower+thorn” structure on the variable nodes represents $L \cdot (1 - x_k) + w - w \cdot (x_i \cdot x_j \cdot (1 - x_k))$, where $L > w + 1$ is a large real number. By constructing gadget graphs that cancel out the lower

order terms as shown before, we arrive at a lifted graphical representation of the positive nonlinear term.

Procedure *decompose-polynomial* on line 3 of Algorithm 1 takes as input the polynomial p_i associated with a constraint f_i , constructed in stage 1, and returns its lifted representation G_{CCG_i} , where $X_i = \mathbf{x}^{f_i}$, Y_i and Z_i are the set of auxiliary variables introduced by the procedure, E_i is the set of edges between the G_{CCG_i} graph nodes, and w_i is the set of weights associated with the variables in X_i, Y_i , and Z_i . For a variable $v_i \in X_i \cup Y_i \cup Z_i$, a unary constraint f_{v_i} in $\mathbf{F}_{CCG_i}$ is

$$f_{v_i}(v_i) = \begin{cases} w_i, & \text{if } v_i = 1, \\ 0, & \text{if } v_i = 0. \end{cases} \quad (3)$$

For each edge $\{v_i, v_j\}$ in E_i , a constraint $f_{\{v_i, v_j\}}$ in $\mathbf{F}_{CCG_i}$ is defined as

$$f_{\{v_i, v_j\}}(v_i, v_j) = \begin{cases} \infty, & \text{if } v_i = v_j = 0, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

For a CCG gadget graph G_{CCG_i} , X_i contains nodes that correspond to decision variables, Z_i contains the nodes with weight L (if any), and Y_i contains the other nodes. At the end of this stage, each agent $a_i \in \mathbf{A}$ controls the set of decision variables in $\mathbf{X}_i$ and the set of auxiliary variables $\cup_{f_j \in \mathbf{F}_i} (Y_j \cup Z_j)$ for all constraints $f_j \in \mathbf{F}_i$ controlled by agent a_i .

3. Merging Gadget Graphs into a CCG Finally, the CCG-Max-Sum agents construct the CCG by merging their gadget graphs G_{CCG_i} . This stage is done incrementally. Every time an agent builds a new gadget graph, it (1) updates its internal graphical representation to include the auxiliary variables introduced by the construction, and (2) increases the weight associated with the agent's variables. Each agent a_i sends to its neighbor a_j all unary constraints in $\mathbf{F}_{CCG_i}$ involving variable v_j controlled by agent a_j (i.e., $\alpha(v_j) = a_j$) (lines 4-5). When an agent receives a new unary constraint f which involves one of its decision variables v_i , it increases the weight associated with the constraint ($f_{v_i}(v_i)$) by the value $f_{v_i}(1)$ (line 6). The communication structure of the underlying DCOP does not vary after the CCG construction. If an agent a_i is a neighbor of an agent a_j in the constraint graph of the original DCOP, then a_i is also a neighbor of a_j in the lifted DCOP representation.

Figure 4 shows the construction of the CCG associated with our example DCOP of Fig. 1. There are three unary and three binary constraints. Their lifted graphical representations are shown next to them. Every node in the CCG is given a weight equal to the sum of the individual weights of the nodes in the CCG gadget graphs. Computing the MWVC for the CCG yields an optimal solution for the DCOP: If variable $x_i \in \mathbf{X}$ is in the MWVC, then it is assigned the value 1 in the DCOP, otherwise it is assigned the value 0.

4.2 Message-Passing Phase

Once the CCG has been constructed, the agents start the message-passing phase to find a node cover with a small total weight. The message-passing scheme is

similar to that of Max-Sum: During each iteration, each agent waits to receive all messages from its neighbors, updates the current values (beliefs) for the variables it controls, computes the messages to send to its neighbors based on its new beliefs, and sends these to all of its neighbors. Here, we adapt the algorithm presented in [25] (see Algorithm 1). Differently from Max-Sum, where each function node exchanges messages with its neighboring variable nodes, and each variable node exchanges messages with its neighboring function nodes, in CCG-Max-Sum, the messages are exchanged between (decision and auxiliary) variables nodes in the CCG. The message $\mu_{u \rightarrow v}$ sent by a variable u to a variable v in iteration i is:

$$\mu_{u \rightarrow v}^i = \max \left\{ w_u - \sum_{t \in N(u) \setminus \{v\}} \mu_{t \rightarrow u}^{i-1}, 0 \right\}, \quad (5)$$

where w_u is the weight associated with variable u , and $N(u)$ is the set of neighboring variables of variable u in the CCG. Equation (5) is derived from Eqs. (1) and (2) using an approach similar to that in [25]. These steps are shown on lines 7–11 of Algorithm 1. When the termination condition (e.g., a convergence criteria or a maximum number of iteration) is met, for a node v , if $w_v < \sum_{u \in N(v)} \mu_{u \rightarrow v}^i$, with i being the last iteration of the algorithm, then v is selected into the MWVC; otherwise it is not. A variable is assigned value 1 if its corresponding decision variable node in the CCG is selected into the MWVC; otherwise it is assigned value 0 (lines 12–13).

5 Experimental Evaluation

In this section, we compare the solution costs of CCG-Max-Sum, Max-Sum (executed on the factor graph), and DSA [27], a local search DCOP algorithm. DSA has been shown to outperform several other incomplete DCOP algorithms [3, 10] and performs similarly to several Max-Sum variants, including Max-Sum_ADVP [29], which has been shown not to benefit from damping [3], where message values are modified to follow a weighted moving average process. We also analyze the effect of using the NT reduction [18], which solves a polynomial-time relaxation of the MWVC to expose optimal assignments to sets of variables, in conjunction with CCG-Max-Sum (denoted by CCG-Max-Sum-k). The NT reduction is executed as a preprocessing centralized step.³ We use DSA-C with $p = 0.6$, where agents decide probabilistically if to select a local-non-worsening assignment, and adopt a damping strategy with weight 0.7 in all Max-Sum variants [3].

We evaluate all algorithms on *federated social network problems*—an application domain that we introduce below—and on random minimization Boolean DCOPs over three classical networks topologies [11]: *grid networks*, *scale-free networks*, and *random networks*, to cover both structured and unstructured problems. We implement all algorithms within an anytime framework, as proposed

³ Its runtime is comparable to that of one iteration of CCG-Max-Sum, which in turn takes 0.035 seconds on average in our experiments.

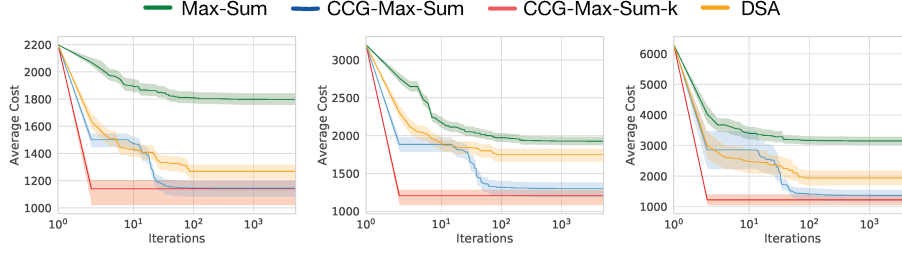


Fig. 5: FSN based on Twitter network data: 100 agents (left), 500 agents (center), and 1000 agents (right).

in [28], where the agents memorize the best solution found up to the current iteration. All results are averages of 30 runs.

Federated Social Networks To address the privacy concerns raised in modern centralized social networks, open-source communities have developed decentralized social networks, such as *Diaspora*, *GNU Social*, and *pump.io* [23]. A *federated social network* (FSN) adopts a decentralized structure by allowing each user or group of users to maintain its server and communicating using a common inter-server protocol. In an FSN, multiple servers are used to store the information of the social network users. A server a_i fetches information from a server a_j if a user in a_i follows a user in a_j [23]. Qualitatively speaking, there are two fetching strategies: **freq-fetch**, that fetches frequently and caches less information, and **more-cache**, that fetches less frequently and caches more fetched information. Each strategy has its own advantages and disadvantages: **freq-fetch** incurs higher bandwidth costs but lower storage costs, while **more-cache** incurs lower bandwidth costs but higher storage costs. Since **freq-fetch** incurs bandwidth costs for both servers, this strategy takes effect between two servers only if both have the strategy **freq-fetch**.

We model the relationship between the costs and fetching strategies as a DCOP. The choice of strategy of each server a_i (which is modeled as an agent) is a variable x_i . $x_i = 1$ implies **freq-fetch**, and $x_i = 0$ implies **more-cache**.

The binary $f(x_i, x_j)$ cost functions capture the storage and bandwidth costs for servers a_i and a_j . A user in a_i following a user in a_j and a user in a_i and one in a_j following each other are modeled, respectively, as

$$\begin{cases} \alpha_{ij}(c_i^b + c_j^b), & \text{if } x_i = x_j = 1 \\ \alpha_{ij}c_i^s, & \text{otherwise} \end{cases} \quad \begin{cases} (\alpha_{ij} + \alpha_{ji})(c_i^b + c_j^b), & \text{if } x_i = x_j = 1 \\ \alpha_{ij}c_i^s + \alpha_{ji}c_j^s, & \text{otherwise,} \end{cases}$$

where c_i^b and c_i^s denote the unit bandwidth and unit storage costs of agent a_i , respectively, and α_{ij} is the amount of information that a_i fetches from a_j .

We model an FSN based on *Twitter network data* [4], which describe a graph whose nodes model Twitter users. There is a link between two nodes if at least one of the corresponding users follows the other one. The graph contains 456,626 nodes and 14,855,842 edges. We map the Twitter network to an FSN graph G . Its nodes represent the FSN servers and are constructed as follows. We first

randomly assign one distinct Twitter user to each node in G . Then, we associate each remaining user u to a node of G with a probability proportional to the number of followers user u has in the corresponding server. We add an edge (a_i, a_j) to G if there exist a user in a_i and a user in a_j such that at least one of them follows the other one. The costs c_i^b and c_i^s are generated by sampling from the discrete uniform distribution $U(1, 10)$, and all weights α_{ij} are the number of users in a_i following users in a_j .

Figure 5 illustrates the anytime behavior of the algorithms on FSN problems with 100 (left), 500 (center), and 1000 (right) agents. The shaded region around each line describes the confidence interval of the solution costs reported by each algorithm. The plots use a log-10 scale for the x-axis. The algorithms in order of their solution costs (from highest to lowest) tend to be: Max-Sum, DSA, and both CCG-Max-Sum variants. In particular, CCG-Max-Sum-k dominates all other algorithms from the very first iteration.

Random DCOPs We now discuss the solution cost of the algorithms on random minimization Boolean DCOPs. The costs of each assignment to the variables involved in a constraint are generated by sampling from the discrete uniform distribution $U(1, 100)$. For grid networks, we generate two-dimensional 10×10 grids and connect each node with its four nearest neighbors. For scale-free networks, we create an n -node network based on the Barabasi-Albert model [1]. Starting from a connected 2-node network, we repeatedly add a new node, randomly connecting it to two existing nodes. These two nodes are selected with probabilities that are proportional to the numbers of their incident edges. Finally, for random networks, we create an n -node network whose density p_1 produces $\lfloor n(n-1)p_1 \rfloor$ edges. We report experiments on low-density problems ($p_1 = 0.2$) and high density problems ($p_1 = 0.6$) and fix the maximum constraint arity to 4. Constraints of arity 4 and 3, respectively, are generated by merging first all cliques of size 4 and then those of size 3. The other edges are used to generate binary constraints. In each configuration, we verify that the resulting constraint graph is connected and set the number of agents to 100.

The results are similar to the ones on FSN problems: The algorithms in order of their solution costs (from highest to lowest) tend to be: Max-Sum, DSA, and both CCG-Max-Sum variants, except on high-density random networks, where the solution costs of DSA are slightly lower than the ones of the CCG-Max-Sum variants. On grid, scale-free, and low-density random networks, CCG-Max-Sum-k dominates all other algorithms from the first ten iterations. On random networks (Figure 6 (bottom)), the effect of kernelization is negligible and both CCG-Max-Sum variants are thus almost indistinguishable, meaning that both of them dominate all other algorithms on low-density random networks.

Thus, our experiments suggest that CCG-Max-Sum has strong advantages on grid and scale-free networks, which are important for a large variety of DCOP applications [5, 8, 22].

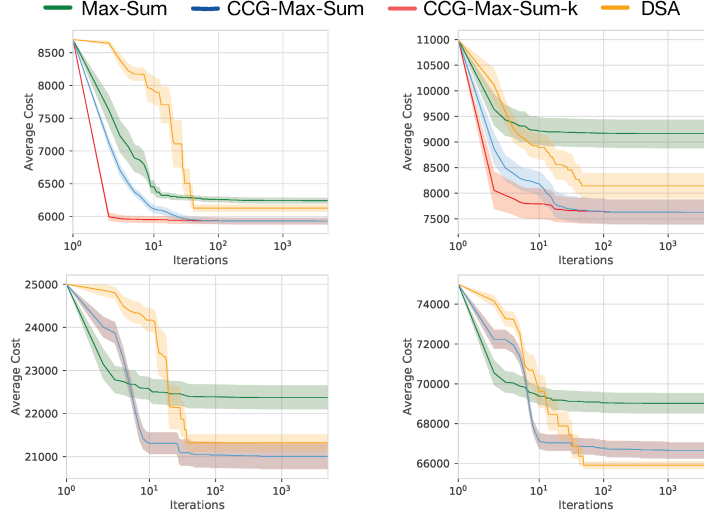


Fig. 6: Grid networks (top left), scale-free networks (top right), low-density random networks ($p_1 = 0.2$) (bottom left), and high-density random networks ($p_1 = 0.6$) (bottom right). The blue and red curves overlap in the last two plots.

x_1	0	0	0	1	1	1
x_2	0	1	2	0	1	2
f_1	0.5	0.6	0.2	0.7	0.3	0.5

Fig. 7: A cost function with a non-Boolean variable. This cost function extends the Boolean cost function in Fig. 1(d) with x_2 being able to take 3 values 0, 1, and 2. The tuples highlighted in red are the parts additional to Fig. 1(d).

6 Discussion: Non-Boolean DCOPs

The construction of the CCG for CCG-Max-Sum can be extended to DCOPs with non-Boolean domains [14] as outlined in the following.

1. Expressing Constraints as Polynomials For a cost function with non-Boolean variables, this step outputs polynomials of degrees at least 2 instead of polynomials of degree 1. The degree of each variable equals its domain size - 1. Fig. 7 shows an example cost function. Similar to Boolean DCOPs, a polynomial of the following form can be used to characterize this cost function:

$$p_1(x_1, x_2) = c_{00} + c_{01}x_1 + c_{10}x_2 + c_{11}x_1x_2 + c_{20}x_2^2 + c_{21}x_1x_2^2.$$

Here, the coefficients $c_{00}, c_{01}, c_{10}, c_{11}, c_{20}$, and c_{21} can be computed by solving a system of linear equations, where each equation corresponds to an entry in the constraint table, using standard Gaussian Elimination. In our example:

$$\begin{array}{lll} p_1(0, 0) = 0.5 & p_1(0, 1) = 0.6 & p_1(0, 2) = 0.2 \\ p_1(1, 0) = 0.7 & p_1(1, 1) = 0.3 & p_1(1, 2) = 0.5. \end{array}$$

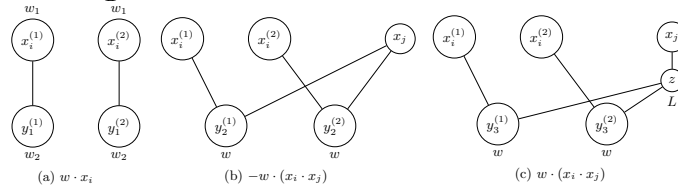


Fig. 8: The lifted graphical representation of terms in a polynomial for linear (a), negative nonlinear (b), and positive nonlinear (c) terms. Here, x_i and x_j have domain sizes 3 and 2, respectively. $x_i^{(1)}$ and $x_i^{(2)}$ are the two nodes representing x_i . We assume that $w > 0$ in (b) and (c). A node has a zero weight if no weight is shown. In (a), w_1 and w_2 satisfy $w_1 - w_2 = w$.

2. Decomposing the Terms of the Polynomials The procedure to construct graph gadgets is similar to Boolean DCOPs, except that each variable x_i with domain $D_{x_i} = \{0, 1, \dots, |D_{x_i}| - 1\}$ is now represented by $|D_{x_i}| - 1$ nodes in the gadget graph. The value of x_i in the to-be-determined optimal solution equals the number of nodes representing x_i in the computed MWVC. Fig. 8 illustrates the lifted representation of linear terms, negative non-linear terms, and positive non-linear terms. It is not hard to verify that Fig. 8 (a-c) represent $w \cdot x_i$, $2w - w \cdot (x_i \cdot x_j)$ and $L \cdot (1 - x_j) + 2w - w \cdot (x_i \cdot (1 - x_j))$, respectively.

3. Merging Gadget Graphs into a CCG Similar to Boolean DCOPs, a CCG can be constructed by merging all nodes corresponding to the same variable. The size of the CCG increases only polynomially in the domain sizes.

Since the agents need to control the value of several CCG nodes, the local solving process, in which an agent decide the value assignments for each of the variables it controls, can be handled with a framework similar to that presented in [7], which was shown highly effective for solving multi-variable agent DCOPs.

7 Conclusions

In this paper, we adapted the *Constraint Composite Graph (CCG)* graphical representation encoding for Distributed Constraint Optimization Problems (DCOPs). The CCG provides a framework for exploiting simultaneously the graphical structure of the agent interaction process as well as the numerical structure of the constraints of a DCOP instance. We use this representation to introduce CCG-Max-Sum, a novel incomplete DCOP algorithm which extends Max-Sum by executing the distributed message passing phase on the CCG.

Compared to a version of Max-Sum which is executed on factor graphs and other incomplete DCOP algorithms, CCG-Max-Sum finds solutions of better quality within fewer iterations on several DCOP benchmarks. While this paper introduced an inference-based algorithm operating on the CCG of a DCOP, we believe that CCGs can also be exploited with other classes of DCOP algorithms.

Future directions include extending the experiments evaluation to DCOPs with non-Boolean variables and applying CCG-Max-Sum to problems with hard constraints (e.g., constraints whose costs are either 0 or ∞), since many types of

hard constraints can be simplified during the construction of the CCG, resulting in smaller problems.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. *Science* 286(5439), 509–512 (1999)
2. Chlebík, M., Chlebíková, J.: Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics* 156(3), 292–312 (2008)
3. Cohen, L., Zivan, R.: Max-sum revisited: The real power of damping. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 1505–1507 (2017)
4. De Domenico, M., Lima, A., Mougél, P., Musolesi, M.: The anatomy of a scientific rumor. *Scientific reports* 3, 2980 (2013)
5. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 639–646 (2008)
6. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research* 61, 623–698 (2018)
7. Fioretto, F., Yeoh, W., Pontelli, E.: Multi-variable agents decomposition for DCOPs. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. pp. 2480–2486 (2016)
8. Fioretto, F., Yeoh, W., Pontelli, E.: A multiagent system approach to scheduling devices in smart homes. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 981–989 (2017)
9. Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., Ranade, S.: A DCOP approach to the economic dispatch with demand response. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 981–989 (2017)
10. Hoang, K.D., Fioretto, F., Yeoh, W., Pontelli, E., Zivan, R.: A large neighboring search schema for multi-agent optimization. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. pp. 688–706 (2018)
11. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 133–140 (2010)
12. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47(2), 498–519 (2001)
13. Kumar, T.K.S.: A framework for hybrid tractability results in Boolean weighted constraint satisfaction problems. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. pp. 282–297 (2008)
14. Kumar, T.K.S.: Lifting techniques for weighted constraint satisfaction problems. In: *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM)* (2008)
15. Kumar, T.K.S.: Kernelization, generation of bounds, and the scope of incremental computation for weighted constraint satisfaction problems. In: *Proceedings of*

- the International Symposium on Artificial Intelligence and Mathematics (ISAIM) (2016)
16. Maheswaran, R., Pearce, J., Tambe, M.: Distributed algorithms for DCOP: A graphical game-based approach. In: Proceedings of the Conference on Parallel and Distributed Computing Systems (PDCS). pp. 432–439 (2004)
 17. Modi, P., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1–2), 149–180 (2005)
 18. Nemhauser, G.L., Trotter, L.E.: Vertex packings: Structural properties and algorithms. *Mathematical Programming* 8(1), 232–248 (1975)
 19. Pearce, J., Tambe, M.: Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1446–1451 (2007)
 20. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1413–1420 (2005)
 21. Petcu, A., Faltings, B., Mailler, R.: PC-DPOP: A new partial centralization algorithm for distributed optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 167–172 (2007)
 22. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 468–474 (2016)
 23. Silva, G., Reis, L., Terceiro, A., Meirelles, P., Kon, F.: Implementing federated social networking: Report from the trenches. In: Proceedings of the International Symposium on Open Collaboration (OpenSym). pp. 8:1–8:10 (2017)
 24. Vinyals, M., Rodríguez-Aguilar, J., Cerquides, J.: Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Journal of Autonomous Agents and Multi-Agent Systems* 22(3), 439–464 (2011)
 25. Xu, H., Kumar, T.K.S., Koenig, S.: The Nemhauser-Trotter reduction and lifted message passing for the weighted CSP. In: International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR). pp. 387–402 (2017)
 26. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research* 38, 85–133 (2010)
 27. Zhang, W., Wang, G., Xing, Z., Wittenberg, L.: Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161(1–2), 55–87 (2005)
 28. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence* 212, 1–26 (2014)
 29. Zivan, R., Parash, T., Naveh, Y.: Applying max-sum to asymmetric distributed constraint optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 432–439 (2015)
 30. Zivan, R., Yedidsion, H., Okamoto, S., Grinton, R., Sycara, K.: Distributed constraint optimization for teams of mobile sensing agents. *Journal of Autonomous Agents and Multi-Agent Systems* 29(3), 495–536 (2015)