Improving DPOP with Branch Consistency for Solving Distributed Constraint Optimization Problems

Ferdinando Fioretto[1,2], Tiep Le[1], William Yeoh[1], Enrico Pontelli[1], and Tran Cao Son[1]

[1] Department of Computer Science, New Mexico State University, USA
[2] Department of Mathematics and Computer Science, University of Udine, Italy
{ffiorett,tile,epontell,wyeoh,tson}@cs.nmsu.edu

Abstract. The DCOP model has gained momentum in recent years thanks to its ability to capture problems that are naturally distributed and cannot be realistically addressed in a centralized manner. Dynamic programming based techniques have been recognized to be among the most effective techniques for building complete DCOP solvers (e.g., DPOP). Unfortunately, they also suffer from a widely recognized drawback: their messages are exponential in size. Another limitation is that most current DCOP algorithms do not actively exploit hard constraints, which are common in many real problems. This paper addresses these two limitations by introducing an algorithm, called BrC-DPOP, that exploits arc consistency and a form of consistency that applies to paths in pseudo-trees to reduce the size of the messages. Experimental results shows that BrC-DPOP uses messages that are up to one order of magnitude smaller than DPOP, and that it can scale up well, being able to solve problems that its counterpart can not.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) are constraint optimization problems where variables and constraints are distributed among a group of agents, and where each agent can only interact with agents that share a common constraint [20,24,30]. As a result, agents need to coordinate their value assignments to maximize the overall sum of resulting constraint utilities and lead to an optimal solution of the optimization problem. DCOPs provide an elegant and effective modeling of problems that have a distributed nature, and where a collective is trying to achieve a globally optimal solution within the confines of the localized communication. Researchers have used DCOPs to model various distributed optimization problems, such as meeting scheduling [19,34], resource allocation [8,33], and power network management problems [16].

In recent years, we have witnessed a growing interest towards DCOPs, with the development of a number of complete and incomplete distributed algorithms. A number of implementations have been proposed and are publicly available [18,28,7]. The majority of the existing DCOP algorithms can be placed in one of three classes. *Search-based* algorithms perform a distributed search over the space of solutions to determine optimum [20,9,32]. *Inference-based* algorithms, on the other hand, make use

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 307–323, 2014.

of techniques from dynamic programming to propagate aggregate information among agents [24,8]. Finally, *sampling-based* algorithms rely on sampling applied to the overall search space [23,22]. Of these methods, the *Distributed Pseudo-tree Optimization Procedure*[1] (DPOP) [24] is one of the most efficient DCOP solvers; DPOP has also been extended in several ways to enhance its performance and capabilities (e.g., O-DPOP and MB-DPOP trade off memory requirement for longer runtimes [26,27], A-DPOP trades off solution optimality for shorter runtimes [25], SS-DPOP trades off runtime for increased privacy [10], H-DPOP exploits hard constraints for smaller runtimes [17], and DPOP with function filtering exploits utility bounds for smaller runtimes [3]).

This paper proposes a novel variant of DPOP, called *Branch-Consistency DPOP* (BrC-DPOP), that takes advantage of hard constraints present in the problem to prune the search space. BrC-DPOP introduces a new form of consistency, called *branch consistency*, which can be viewed as a weaker version of path consistency [21] tailored to variables ordered in a pseudo-tree, and where each agent can only communicate with neighboring agents. The effect of enforcing this consistency in DPOP is the ability to actively use hard constraints (either explicitly provided in the problem specification or implicitly described in utility tables) to prune the search space and to reduce the size of the utility tables exchanged among agents.

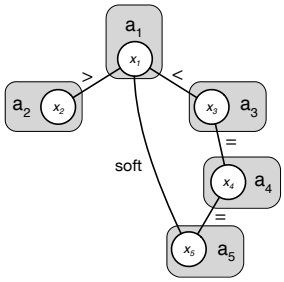
2 Background

2.1 Distributed Constraint Optimization Problems (DCOPs)

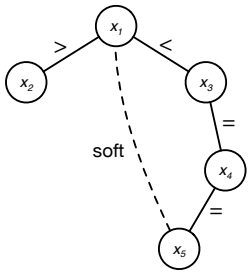
A *DCOP* [20,24,30] is defined by a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of *variables*; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite *domains*, where D_i is the domain of variable x_i; $\mathcal{F} = \{f_1, \ldots, f_e\}$ is a set of *utility functions* (also called *constraints*), $f_i : \times_{x_j \in scope(f_i)} D_j \mapsto \mathbb{N} \cup \{0, -\infty\}$, specifying the utility of each combination of values to the variables in the *scope* of the constraint (where $scope(f_i) \subseteq \mathcal{X}$); $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of *agents*; $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. In this paper, we will focus on *unary* and *binary* constraints. For simplicity, we will refer to unary constraints as f_{ii} and binary constraints as f_{ij} to denote the fact that their scope is $\{x_i\} \subseteq \mathcal{X}$ and $\{x_i, x_j\} \subseteq \mathcal{X}$, respectively. We also assume that each agent has exactly one variable mapped to it. Thus, we will use the terms "variable" and "agent" interchangeably. This is a common assumption in the DCOP literature as there exist pre-processing techniques that transform a general DCOP into our more restrictive DCOP [31,4]. A solution is a value assignment for a subset of variables. Its utility is the evaluation of all utility functions on it. A solution is *complete* if it is a value assignment for all variables in $\mathcal{X}$. The goal is to find a utility-maximal complete solution.

Each constraint in $\mathcal{F}$ can be either *hard*, indicating that some value combinations result in a utility of $-\infty$ and must be avoided, or *soft*, indicating that all value combinations result in a finite utility and need not be avoided. We use $\mathbf{H}_i$ and $\mathbf{S}_i$ to denote the set of hard and soft constraints, respectively, whose scope includes x_i. With a slight abuse of notation, we will also often view a unary constraint f_{ii} as a subset of D_i, defined as $f_{ii} = \{(u) \in D_i \mid f_{ii}(u) \neq -\infty\}$, and a binary constraint f_{ij} as a subset of $D_i \times D_j$, defined as $f_{ij} = \{(u, v) \in D_i \times D_j \mid f_{ij}(u, v) \neq -\infty\}$.

[1] This algorithm has also been referred to as Dynamic Programming Optimization Protocol.



(a) Constraint Graph



(b) Pseudo-tree

x_1	x_5	Utilities
0	0	20
0	1	8
0	2	10
0	3	3
...		
3	3	2

(c) Constraint Table

Fig. 1. Example DCOP

Table 1. Example UTIL Phase Computations of a_5

x_1	x_4	Utilities
0	0	max(20+0, 8-∞, 10-∞, 3-∞) = 20
0	1	max(20-∞, 8+0, 10-∞, 3-∞) = 8
0	2	max(20-∞, 8-∞, 10+0, 3-∞) = 10
0	3	max(20-∞, 8-∞, 10-∞, 3+0) = 3
...		...

A *constraint graph* visualizes a DCOP, where nodes correspond to the variables and the edges connect pairs of variables in the scope of the same utility function. A *DFS pseudo-tree* arrangement has the same nodes and edges as the constraint graph and satisfies two conditions: *(i)* there is a subset of edges (*tree edges*) that form a rooted tree, and *(ii)* two variables in the scope of the same utility function appear in the same branch of the tree. The other edges are called *backedges*. Tree edges connect parent-child nodes, while backedges connect a node with its pseudo-parents and pseudo-children. We also use the following notation: C_i, PC_i, P_i, and PP_i refer to the set of children, pseudo-children, parent, and pseudo-parents of agent a_i, respectively; and $sep(a_i)$ refers to the *separator* of agent a_i, which is the set of ancestor agents that are constrained with agent a_i or one of its descendant agents.

Figure 1(a) shows the constraint graph of a simple DCOP with five agents, a_i, with $i = 1, \ldots, 5$, each owning exactly one variable x_i. The domain of each variable is the set $\{0, 1, 2, 3\}$. Figure 1(b) shows one possible pseudo-tree for the problem, where the agent a_1 has one pseudo-child, a_5 (the dotted line is a backedge). Figure 1(c) describes few value combinations of the utility function associated with the constraint f_{15}.

2.2 Distributed Pseudo-Tree Optimization Procedure (DPOP)

DPOP [24] has the following three phases:

(1) The first phase is the pseudo-tree generation phase, realized through an existing distributed pseudo-tree construction algorithm, like Distributed DFS [15].

(2) The second phase is the UTIL propagation phase, where each agent, starting from the leaves of the pseudo-tree, computes the optimal sum of utilities in its subtree for

each value combination of variables in its separator. The agent does so by summing the utilities of its constraints with the variables in its separator and the utilities in the UTIL messages received from its children agents, and then projecting out its own variables by optimizing over them. In our example problem, agent a_5 computes the optimal utility for each value combination of variables x_1 and x_4, as shown in Table 1, and sends the utilities to its parent agent a_4 in a UTIL message. Such a table consists of $4^3 = 64$ utilities before projecting out its variable x_5, and $4^2 = 16$ utilities after the projection. The value 0 ($-\infty$) represents the utility for the hard constraint f_{45} for values of x_4, x_5 that satisfy (do not satisfy) it. When the root agent a_1 receives the UTIL message from each of its children, it computes the maximum utility of the entire problem.

(3) The third phase is the VALUE propagation phase, where each agent, starting from the root of the pseudo-tree, determines the optimal value for its variable. The root agent does so by choosing the value of its variable from its UTIL computations—selecting the value with the maximum utility. It sends the selected value to its children in a VALUE message. Each agent receiving a VALUE message will determine the value for its variable producing the maximum utility given the variable assignments (of the agents in its separator) indicated in the VALUE message. Once determined, such assignment is further propagated to the children via VALUE messages.

3 Branch-Consistent DPOP (BrC-DPOP)

3.1 Preliminaries

Definition 1. *The* consistency graph *of a DCOP* $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$ *is* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *where* $\mathbf{V} = \{(i,k) \mid x_i \in \mathcal{X}, k \in D_i\}$ *and* $\mathbf{E} = \{\langle (i,r),(j,c)\rangle \mid r \in D_i, c \in D_j, f_{ij} \in \mathcal{F}, (r,c) \in f_{ij}\}$.

Definition 2. *Given a pseudo-tree associated with a DCOP problem instance, we define a* linear ordering $\prec$ *on its variables:* $x_i \prec x_j$ *iff* $x_j \in P_i$. *Similarly,* $x_i \succ x_j$ *iff* $x_j \in C_i$. *We denote with* $\preceq$ *(and* $\succeq$*) the reflexive closure of* $\prec$ *(and* $\succ$*), and with* $\overset{*}{\prec}$ *(and* $\overset{*}{\succ}$, $\overset{*}{\preceq}$, $\overset{*}{\succeq}$*) the transitive closure of* $\prec$ *(and* $\succ$, $\preceq$, $\succeq$ *).*

Definition 3. *A pair of values* $(r,c) \in D_i \times D_j$ *of two variables* x_i, x_j *that share a constraint* f_{ij} *is* branch consistent (BrC) *iff for any sequence of variables* $(x_i = x_{k_1}, \ldots, x_{k_m} = x_j)$, *such that* $f_{k_p k_q} \in \mathcal{F}$, *where* $p \leq q \leq p+1$, *and* $x_{k_1} \preceq \cdots \preceq x_{k_m}$, *there exists a tuple of values* $(r = v_{k_1}, \ldots, v_{k_m} = c)$ *such that* $v_{k_q} \in D_{k_q}$ *and* $(v_{k_p}, v_{k_q}) \in f_{k_p k_q}$, *for each* $1 \leq q \leq m$ *and* $p \leq q \leq p+1$.

Definition 4. *A DCOP is* branch consistent (BrC) *iff for any pair of variables* (x_i, x_j) *with* $x_i \preceq x_j$ *and any* $(u,v) \in f_{ij}$, (u,v) *is branch consistent.*

Definition 5. *Given a DCOP, the* Value Reachability Matrix (VRM) M_{ij} *of two variables* x_i *and* x_j *of* $\mathcal{X}$, *with* $x_i \overset{*}{\preceq} x_j$, *is a binary matrix of size* $D_i \times D_j$, *where* $M_{ij}[r,c]=1$ *iff there exists at least one sequence of variables* $(x_i = x_{k_1}, \ldots, x_{k_m} = x_j)$, *such that* $x_{k_1} \preceq \cdots \preceq x_{k_m}$, *and a tuple of values* $(r = v_{k_1}, v_{k_2}, \ldots, v_{k_m} = c)$ *such that* $v_{k_p} \in D_{k_p}$ *and* $(v_{k_p}, v_{k_q}) \in f_{k_p k_q}$, *for each* $1 \leq q \leq m$ *and* $p \leq q \leq p+1$.

Proposition 1. *For each variable x_i, x_j, and x_k, the* regular product *of two VRMs M_{ik} and M_{kj} is a VRM $M_{ij} = M_{ik} \times M_{kj}$, where each entry (r, c) of M_{ij} is given by*

$$M_{ij}[r, c] = \min \left\{ 1, \sum_{l=1}^{|D_k|} M_{ik}[r, l] \cdot M_{kj}[l, c] \right\}$$

Proposition 2. *For each variable x_i and x_j, the* entrywise product *of two VRMs M_{ij} and $\hat{M}_{ij}$ is a VRM $M'_{ij} = M_{ij} \circ \hat{M}_{ij}$, where each entry (r, c) of M'_{ij} is given by*

$$M'_{ik}[r, c] = M_{ij}[r, c] \cdot \hat{M}_{ij}[r, c]$$

Definition 6. *Given a VRM M_{ij}, a pair of values (r,c) is a* valid *pair iff $M_{ij}[r, c] = 1$.*

Definition 7. *If $f_{ij} \in \mathcal{F}$, then M_{ij} is* branch consistent (BrC) *iff all its valid pairs are branch consistent. If $f_{ij} \notin \mathcal{F}$, then M_{ij} is* branch consistent *iff it is a regular product of branch consistent VRMs.*

3.2 High-Level Algorithm Description

Let us now illustrate the high-level structure of BrC-DPOP on the example DCOP shown in Figure 1. BrC-DPOP consists of the following phases:

- **Pseudo-tree Generation Phase:** This phase is identical to that of DPOP.
- **Path Construction Phase:** In this phase, each agent builds the VRMs associated with the constraints involving its variables along with the structures describing the paths between pseudo-parents and pseudo-children. Figure 2(a) shows the VRMs (in a consistency graph representation); we do not show the soft constraint between variables x_1 and x_5 as it allows every value combination of the two variables.
- **Arc Consistency Enforcement Phase:** In this phase, the agents enforce arc consistency in a distributed manner. At the end of this phase, each agent has the updated VRMs shown in Figure 2(b). Arc consistency causes the removal of exactly two values from the domain of each variable of the DCOP: values 0 and 3 from D_1, 0 and 1 from D_2, and 2 and 3 from D_3, D_4, and D_5.
- **Branch Consistency Enforcement Phase:** In this phase, the agents enforce branch consistency in a distributed manner. In our example, branch consistency needs to be enforced for the pairs of values of variables x_1 and x_5 only. The values for all other pairs of variables are already branch consistent. Agent a_1 starts this process by sending a message containing VRM M_{11} to its child a_3 (since a_5 is in the subtree rooted at a_3). Once agent a_3 receives the message, it computes the VRM M_{31} by multiplying its VRM M_{31} with the VRM M_{11} just received, and sends a message containing this VRM to its child a_4. Agent a_4 repeats this process by multiplying its VRM M_{43} with the VRM M_{31}, resulting in VRM M_{41}, which it sends to its child a_5. This process repeats until agent a_5 computes the VRM M_{51}, after which it knows its set of reachable values in x_5 for each value in x_1. Figure 2(c) shows the VRMs.
- **UTIL and VALUE Propagation Phases:** This phase is identical to the corresponding UTIL and VALUE propagation phases of DPOP, except that each agent constructs

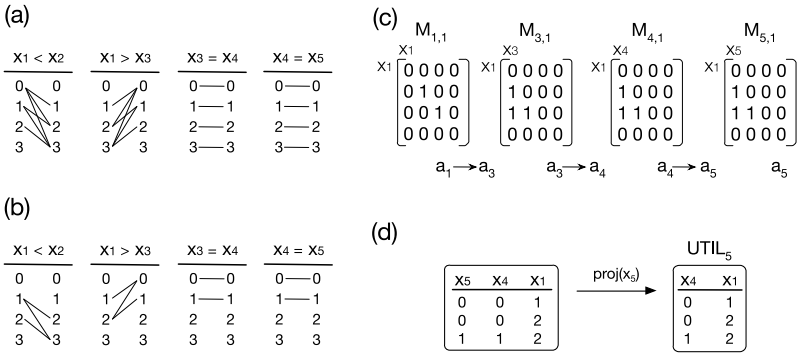


Fig. 2. Example Trace

a UTIL table that contains utilities for each combination of *unpruned* values of variables in its VRMs. In our example, agent a_5 is able to project out its variable x_5 and construct its UTIL table, shown in Figure 2(d). Note that the UTIL table consists of only 3 utilities, both before and after projection. In contrast, DPOP's UTIL table consists of $4^3 = 64$ utilities before projection and $4^2 = 16$ utilities after projection.

3.3 Messages and Data Structures

During the execution of BrC-DCOP, each agent a_i maintains the following data structures, where the first three are used in the arc consistency phase and the last two are used in the branch consistency phase.

- The set of hard constraints $\hat{\mathbf{H}}_i = \{f_{ij} \in \mathbf{H}_i \mid a_i \overset{*}{\preceq} a_j\}$ to check for consistency.
- The set of VRMs $\hat{\mathbf{M}}_i = \{\hat{M}_{ij} \mid f_{ij} \in \mathcal{F}, a_j \overset{*}{\preceq} a_i\}$, which includes the VRMs for each parent and pseudo-parent a_j.
- The flag *fixed*$_i$ for each agent a_i, which is initialized to true. It indicates if agent a_i has reached a fixed point in the arc consistency phase.
- The set of VRMs $\mathbf{M}_i = \{M_{ij} \mid a_j \in sep(a_i)\}$, which includes the VRMs for each separator agent a_j.
- The set of paths $\text{PATHS}_i = \{(a_s \overset{a_j}{\rightsquigarrow} a_d) \mid a_j \in C_i,\ a_s \overset{*}{\succeq} a_i \succ a_j \overset{*}{\succeq} a_d\}$, which the agent uses to send updated VRMs in the branch consistency phase. Each path $(a_s \overset{a_j}{\rightsquigarrow} a_d)$ indicates that there is a branch in the pseudo-tree from a_s to a_d that passes through a_i and its child a_j. This data is needed by agent a_i to know which child it should send its updated VRM to, if the VRM originated from agent a_s. For example, in our example trace, agent a_1 knows to send its VRM to its child a_3 and not a_2. To preserve privacy, the information about the destination agent a_d can be omitted from each path. Each agent thus maintains only $(a_s \overset{a_j}{\rightsquigarrow} ?)$, which is sufficient to ensure correctness.

In addition to the UTIL and VALUE messages used in the UTIL and VALUE propagation phases, each agent a_i uses the following types of messages, where the first

Algorithm 1. BRC-DPOP

1 PSEUDO-TREE-GENERATION-PHASE( )
2 PATH-CONSTRUCTION-PHASE( )
3 AC-PROPAGATION-PHASE( )
4 BRC-PROPAGATION-PHASE( )
5 UTIL-AND-VALUE-PHASES( )

Procedure Path-Construction-Phase( )

6 **foreach** $a_p \in PP_i$ **do**
7 $\text{PATHS}_i \leftarrow \text{PATHS}_i \cup (a_p \overset{NULL}{\rightsquigarrow} ?)$
8 **if** $C_i \neq \emptyset$ **then**
9 **while** *not received all* $PATH_c^{\uparrow}(\cdot)$ *from each* $a_c \in C_i$ **do**
10 **if** *receive* $PATH_c^{\uparrow}(a_s)$ *from* $a_c \in C_i$ **then**
11 $\text{PATHS}_i \leftarrow \text{PATHS}_i \cup (a_s \overset{a_c}{\rightsquigarrow} ?)$
12 **foreach** $a_s \neq a_i$ *such that* $(a_s \overset{a_c}{\rightsquigarrow} ?) \in \text{PATHS}_i$ **do**
13 Send $PATH_i^{\uparrow}(a_s)$ to P_i
14 **if** $PATH_i^{\uparrow}(\cdot)$ *has not been sent to* P_i **then**
15 Send $PATH_i^{\uparrow}(NULL)$ to P_i

two are used in the arc consistency phase, while the last two in the branch consistency phase:[2]

- $AC_i^{\uparrow}(D'_j, \mathit{fixed}_i)$, which is sent from an agent a_i to an agent $a_j \overset{*}{\succ} a_i$ such that $f_{ij} \in \mathbf{H}_i$. It contains a copy of the domain of the variable x_j, D'_j, updated with the changes caused by the propagation of the constraints in $\hat{\mathbf{H}}_i$, and a flag, fixed_i, which denotes whether changes have occurred in the domain of some variable in the subtree rooted at a_i during the last iteration of the $AC^{\uparrow}$ messages.
- $AC_i^{\downarrow}(D_i)$, which is sent from an agent a_i to the agents $a_j \overset{*}{\prec} a_i$ such that $f_{ij} \in \mathbf{H}_i$. It contains a copy of the domain of the variable x_i, D_i, updated with the changes caused by the propagation of the constraints in $\hat{\mathbf{H}}_i$.
- $PATH_i^{\uparrow}(a_s)$, which is sent from an agent a_i to its parent P_i to inform it that it is part of a tree path in the pseudo-tree between agents a_s and some pseudo-child of a_s.
- $BrC_i^{\downarrow}(M_{is})$, which is used to determine the branch consistent value pairs of x_s and x_i.

3.4 Algorithm Description

Algorithm 1 shows the pseudo-code of BrC-DPOP. It can be visualized as a process composed of 5 phases:

- **Phase 1 - Pseudo-tree Generation Phase:** This phase is identical to that of DPOP, where a pseudo-tree is generated (line 1).

[2] We use the superscript $^{\uparrow}$ to denote the messages being propagated from the leaves of the pseudo-tree to the root, and $^{\downarrow}$ to denote the ones propagated from the root to the leaves.

Procedure AC-Propagation-Phase( )

16 $iteration \leftarrow 0$
17 **repeat**
18 **if** $C_i \neq \emptyset$ **then**
19 Wait until received $AC_c^{\uparrow}(D'_i, fixed_c)$ from each $a_c \in C_i \cup PC_i$ in this iteration
20 **foreach** $AC_c^{\uparrow}(D'_i, fixed_c)$ *received* **do**
21 $D_i \leftarrow D_i \cap D'_i$
22 $\langle \hat{\mathbf{M}}_i, D_i \rangle \leftarrow$ ENFORCEAC$(\hat{\mathbf{H}}_i, \hat{\mathbf{M}}_i, D_i)$
23 $fixed_i \leftarrow \neg$CHANGED$(D_i) \wedge \bigwedge_{a_c \in C_i} fixed_c$
24 Send $AC_i^{\uparrow}(\hat{M}_{ij|j}, fixed_i)$ to each $a_j \in P_i \cup PP_i$
25 **if** $P_i \neq NULL$ **then**
26 Wait until received $AC_p^{\downarrow}(D_p)$ from each $a_p \in P_i \cup PP_i$ in this iteration or received $BrC_p^{\downarrow}(\cdot)$ from parent a_p
27 **if** *received* $BrC_p^{\downarrow}(\cdot)$ *from parent* a_p **then break**
28 **if** $\neg fixed_i$ **then**
29 **foreach** $AC_p^{\downarrow}(D_p)$ *received* **do**
30 update $\hat{M}_{ip}$ with D_p
31 **if** $P_i \neq NULL$ **then**
32 $\langle \hat{\mathbf{M}}_i, D_i \rangle \leftarrow$ ENFORCEAC$(\hat{\mathbf{H}}_i, \hat{\mathbf{M}}_i, D_i)$
33 Send $AC_i^{\downarrow}(D_i)$ to each $a_c \in C_i \cup PC_i$
34 $iteration \leftarrow iteration + 1$
35 **until** $P_i = NULL$ **and** $fixed_i$

- **Phase 2 - Path Construction Phase:** The phase is used to construct the direct paths from each agent to its parent and pseudo-parents. At the start of this phase (line 2), each agent, starting from the leaves of the pseudo-tree, recursively populates its PATHS$_i$ as follows: It saves a path information $(a_p \overset{NULL}{\rightsquigarrow} ?)$ for each of its pseudo-parents a_p (lines 6-7) and sends a $PATH_i^{\uparrow}(a_p)$ message to its parent. When the parent a_i receives a $PATH_c^{\uparrow}$ message from each of its child a_c, it stores the path information in the message in its PATHS$_i$ data structure (lines 9-11). For each path in PATHS$_i$, if it is not the destination agent, then it sends a $PATH_j^{\uparrow}$ message that contains that path to its parent (lines 12-13). If it does not send a $PATH_j^{\uparrow}$ message to its parent, then it sends an empty $PATH_j^{\uparrow}$ message (lines 14-15). These path information will be used in the branch consistency propagation phase later. When the root processes and stores the path information from each of its children, it ends this phase and starts the next AC propagation phase.
- **Phase 3 - Arc Consistency (AC) Propagation Phase:** In this phase, the agents enforce arc consistency in a distributed manner, by interleaving the direction of the AC message flows: from the leaves to the root (lines 18-24) and from the root to the leaves (lines 25-34), until a fixed point is detected at the root (line 35).

  In the first part of this phase (lines 18-24), each agent, starting from the leaves up to the root, recursively enforces the consistency of its hard constraints in $\hat{\mathbf{H}}_i$ (line 22) via the ENFORCEAC procedure, which we implemented using the AC-2001 algorithm [2]. In this process, the agent also updates the VRMs $\hat{\mathbf{M}}_i$ associated with all its constraints $f_{ij} \in \hat{\mathbf{H}}_i$ and its domain D_i to prune all unsupported values. If

Procedure BrC-Propagation-Phase( )

36 **if** $P_i \neq NULL$ **then**
37 Wait until received a $BrC_p^{\downarrow}(M_{ps})$ for each path $(a_s \overset{a_c}{\rightsquigarrow} ?) \in \text{PATHS}_i$ from parent a_p
38 **foreach** $(a_s \overset{a_c}{\rightsquigarrow} ?) \in \text{PATHS}_i$ **do**
39 **if** $a_s = a_i$ **then** $M_{is} \leftarrow \hat{M}_{ii}$
40 **else** $M_{is} \leftarrow \hat{M}_{ip} \times M_{ps}$
41 $M_{is} \leftarrow \hat{M}_{is} \circ M_{is}$
42 **if** $a_c \neq NULL$ **then**
43 Send $BrC_i^{\downarrow}(M_{is})$ to a_c
44 **foreach** $a_c \in C_i$ *that has not been sent a* $BrC_i^{\downarrow}$ *message* **do**
45 Send $BrC_i^{\downarrow}(NULL)$ to a_c

any of its values are pruned, indicating that it has not reached a fixed point, it sets its $fixed_i$ flag to false (line 23). It then sends an $AC_i^{\uparrow}$ message to each of its parent and pseudo-parent a_j, which contains its $fixed_i$ flag as well as a copy of their domains D'_j[3] to notify them about which unsupported values were pruned (line 24). The domain of each agent is updated before enforcing the arc consistency, as soon as it receives all the $AC_i^{\uparrow}$ messages from each of its children and pseudo-children (lines 20-21).

Once the root enforces the consistency of its hard constraints, it checks if it has reached a fixed point (line 28). If it has not, then it starts the next part of this phase, which is similar to the previous one except for the direction of the recursion and the AC message flow (lines 29-34). This phase is carried from the root down to the leaves of the pseudo-tree, and it ends when all the leaves have enforced the consistency of their hard constraints. Then the procedure repeats the first part where the recursion and the AC message flow starts from the leaves again and continues up to the root. This process repeats until a fixed point is reached at the root (line 35), which ends this phase, and starts the next BrC propagation phase.

- **Phase 4 - Branch Consistency (BrC) Propagation Phase:** In this phase, the agents enforce branch consistency in a distributed manner, that is, every pair of values of an agent and its pseudo-parents are mutually reachable throughout every tree path connecting them in the pseudo-tree.

  At the start of this phase, each agent, starting from the root down to the leaves, recursively enforces branch consistency for all tree paths from the root to that agent and sends a $BrC_i^{\downarrow}$ message to each of its children. This message includes the VRM for each path through that child. Once an agent a_i receives all the VRM messages from its parent (lines 36-37), for each path that goes through it (line 38), it creates a new VRM M_{is}. If it is the start of the path, then it sets its VRM $\hat{M}_{ii}$ (line 39), which is arc consistent, as the new VRM M_{is}. Otherwise, it performs the regular product of its VRM $\hat{M}_{ip}$ for the constraint between itself and its parent a_p and the VRM received from the parent M_{ps} and sets it to M_{is} (line 40). Then, to ensure that the VRM M_{is} is branch consistent, it performs the *entrywise product* with the VRM $\hat{M}_{is}$ of its pseudo-parent a_s (line 41). If the agent is the destination of the path, then

[3] In the pseudo-code, we use the notation $\hat{M}_{ij|j}$ to indicate D'_j.

it will use the resulting VRM in the construction of the UTIL messages in the UTIL phase. Otherwise, it will send the VRM to its child agent that is in that path in a $BrC_i^{\downarrow}$ message (lines 42-43). Finally, it will send an empty $BrC_i^{\downarrow}$ to all remaining child agents to ensure that the propagation reaches all the leaves (lines 44-45).

- **Phase 5 - DPOP's UTIL and VALUE Phases:** This phase is identical to the corresponding UTIL and VALUE propagation phases of DPOP, except that each agent constructs a UTIL table that contains utilities for each combination of *unpruned* values of variables in its VRMs.

4 Theoretical Analysis

In this section, we use n, e, and d to denote $|\mathcal{A}|$, $|\mathcal{F}|$, and $\max_{x_i \in \mathcal{X}} |D_i|$, respectively.

Theorem 1. *The AC propagation phase requires $O(nde)$ messages, each of size $O(d)$.*

Proof Sketch: In the worst case, each AC iteration removes exactly one value from one domain. Thus, there are only $O(nd)$ iterations, as there are only $O(nd)$ values among all variables. In each iteration, each agent sends exactly one $AC^{\uparrow}$ message to each parent and pseudo-parent and one $AC^{\downarrow}$ message to each child and pseudo-child. Thus, there are at most $O(e)$ messages sent in each iteration. Each message contains at most the full domain of a variable and the fixed flag, which is $O(d)$. ■

Theorem 2. *The BrC propagation phase requires $O(e)$ messages, each of size $O(d^2)$.*

Proof Sketch: In the BrC propagation phase, each agent sends exactly one $BrC^{\downarrow}$ message to each child, and the phase ends after all the leaves in the pseudo-tree receives a $BrC^{\downarrow}$ message. Each message contains at most a VRM, which is $O(d^2)$. ■

Theorem 3. *The DCOP is arc consistent after the AC propagation phase.*

Proof Sketch: We prove this result by contradiction. Assume that there are $a_i, a_j \in \mathcal{A}$ and $a \in D_i$ such that $\forall b \in D_j$, $(a, b) \notin f_{ij}$. Let $b_1, \ldots, b_m$ be all the (pruned) values in D_j supporting a. We have the following two cases:

- $a_i \in P_j \cup PP_j$. If agent a_j pruned all its values b_r $(1 \leq r \leq m)$ from D_j, then the value a is pruned from the copy of the domain D_i held at a_j ($\hat{\mathbf{M}}_{ji|i}$ will not include the value a) (line 22). When a_i receives an $AC^{\uparrow}$ message from each $a_k \in C_i \cup PC_i$ (including a_j), it updates its own domain with the copy received from each agent (lines 20-21) removing a from D_i and resulting in a contradiction.
- $a_i \in C_j \cup PC_j$. Agent a_j can prune all its values b_r $(1 \leq r \leq m)$ from D_j in the following two ways. In case 1, agent a_i prunes all the values b_r from a copy of D_j during its AC consistency enforcement (line 22), sends up an $AC^{\uparrow}$ message to a_j, and a_j prunes all its values b_r from its D_j. However, in this case, agent a_i would have also pruned value a from its domain, resulting in a contradiction. In case 2, some other agent a_k that shares a constraint f_{kj} with agent a_j prunes all the values b_r from the copy of D_j during its AC consistency enforcement, sends up an $AC^{\uparrow}$ message to

a_j, and a_j prunes all its values b_r from its D_j. In this case, a_j will eventually send an $AC^{\downarrow}$ message to a_i that contains its updated domain without the values b_r. Then, agent a_i will prune value a from its domain in its AC consistency enforcement (line 22), resulting in a contradiction. ■

Theorem 4. *The DCOP is branch consistent after the BrC propagation phase.*

Proof Sketch: We prove by induction on the number of variables in the paths $x_i = x_{k_1}, \ldots, x_{k_m} = x_j$, such that $x_{k_1} \succ \ldots \succ x_{k_m}$.
Base Case ($m = 2$)**:** We know that $x_j \in C_i$ and there is only one path from x_i to x_j via the constraint f_{ij}. Additionally, this constraint is arc consistent because the BrC propagation phase runs after the AC propagation phase. Thus, all the remaining pairs of values in both variables are by definition branch consistent (Definition 3). The VRM M_{ji} is thus branch consistent.
Induction Assumption: Assume that for any $2 \leq q \leq r$ and paths $x_i = x_{k_1}, \ldots, x_{k_q} = x_j$ with $x_{k_1} \succ \ldots \succ x_{k_q}$, there is a VRM M_{ji} that is branch consistent.
Induction Case ($m = r + 1$)**:** We know that the paths from $x_i = x_{k_1}$ to x_{k_r} is branch consistent from the induction assumption. Thus, the VRM $M_{k_r k_1}$ received by $x_{k_{r+1}}$ is branch consistent. Additionally, all the constraints between any x_{k_p} ($1 \leq p \leq r$) and $x_{k_{r+1}}$ are arc consistent because the BrC propagation phase runs after the AC propagation phase. Thus, the VRMs $\hat{M}_{k_{r+1} k_p}$ are also branch consistent.
We now show that the algorithm removes values of $x_{k_{r+1}}$ that are not branch consistent with values of its ancestors in the following two cases:

- For paths that include the constraint between x_r and x_{r+1}, BrC-DPOP takes the regular product (line 40), which removes all inconsistent values.
- For paths that do not include the constraint between x_r and x_{r+1} and, thus, must include the constraint between x_{k_1} and $x_{k_{r+1}}$, BrC-DPOP performs the entrywise product (line 41), which removes all inconsistent values. ■

Theorem 5. *BrC-DPOP is complete and correct.*

Proof Sketch: The completeness and correctness of BrC-DPOP follows from the correctness and completeness of DPOP [24] and the correctness and completeness of the AC and BrC propagation phases (Theorems 1, 2, 3, and 4). ■

Property 1. Both the UTIL and the VALUE phases require $O(n)$ number of messages.

Property 2. The memory requirement of BrC-DPOP is in the worst case exponential in the induced width of the problem for each agent.

Both properties follow trivially from the properties of DPOP since no values are pruned from the AC and BrC propagation phases in the worst case.

5 Related Work

We characterize the approaches that prune values of variables in DCOPs along two general types. Algorithms in the first category *propagates exclusively hard constraints* (BrC-DPOP falls into this category). To the best of our knowledge, the only existing work that

falls into this category is H-DPOP [17], which, like BrC-DPOP, is also an extension of DPOP. The main difference between H-DPOP and BrC-DPOP is that instead of VRMs, each agent a_i in H-DPOP uses *constraint decision diagrams* (CDDs) to represent the space of possible value assignments of variables in its separator set $sep(a_i)$. A CDD is a rooted directed acyclic graph structured by levels, one for each variable in $sep(a_i)$. In each level, a non-terminal node represents a possible value assignment for the associated variable. Each non-terminal node v has a list of successors: one for each value u in the next variable for which the pair (u, v) is satisfied by the constraint between the two variables. As a result of using CDDs, H-DPOP suffers from two limitations: (1) H-DPOP can be slower than DPOP because maintaining and performing join and projection operations on CDD are computationally expensive. In contrast, maintaining and performing operations on VRMs can be faster, which we will demonstrate in the experimental results section later. (2) H-DPOP cannot fully exploit information of hard constraints to reduce the size of UTIL messages. Consider the DCOP instance of Figure 2, where the domains for the variables x_1, x_3, x_4, and x_5 are represented by the set $\{1, \ldots, 100\}$, while the domain for variable x_2 is the set $\{1, 2\}$. In H-DPOP, a_5 is not aware of the constraints $x_1 < x_2$ and $x_1 < x_3$—neither x_2 nor x_3 are in $sep(a_5)$, thus no pruning will be enforced. Its UTIL table will hence contain $100^2 = 10,000$ utilities for each combination of values of x_4 and x_1. This is the same table that DPOP would construct. In contrast, in BrC-DPOP, the domains of x_1 and x_2 will be pruned to $\{1\}$ and $\{2\}$, respectively, and the domains of x_3, x_4, and x_5 to $\{2, \ldots, 100\}$. Therefore, the UTIL table that a_5 sends to a_4 contains $99 \times 1 = 99$ utilities. Aside from these two limitations, a more critical limitation of H-DPOP is its assumption that each agent has knowledge of all the constraints whose scope is a subset of its separator set. This assumption is stronger than the assumptions made by most DCOP algorithms and might cause privacy concerns in some applications. In contrast, BrC-DPOP does not make such assumptions.

Algorithms in the second category *propagates lower and upper bounds*. Researchers have extended search-based DCOP algorithms (e.g., BnB-ADOPT and its enhanced versions [29,12,14]) to maintain soft-arc consistency in a distributed manner [1,13,11]. Such techniques are typically very effective in search-based algorithms as their runtime depends on the accuracy of its lower and upper bounds.

Finally, it is important to note the differences between branch consistency and path consistency [21]. One can view branch consistency as a weaker version of path consistency, where all the variables in a path must be ordered according to the relation $\prec$, and only a subset of all possible paths have to be examined for consistency. Thus, one can view branch consistency as a form of consistency tailored to pseudo-trees, where each agent can only communicate with neighboring agents.

6 Experimental Results

We implemented a variant of BrC-DPOP, called AC-DPOP, that enforces arc consistency only in order to assess the impact of the branch consistency phase in BrC-DPOP. Moreover, in order to be as comprehensive as possible in our evaluations, we also implemented a variant of H-DPOP called PH-DPOP, which stands for Privacy-based H-DPOP, that restricts the amount of information that each agent can access to the amount

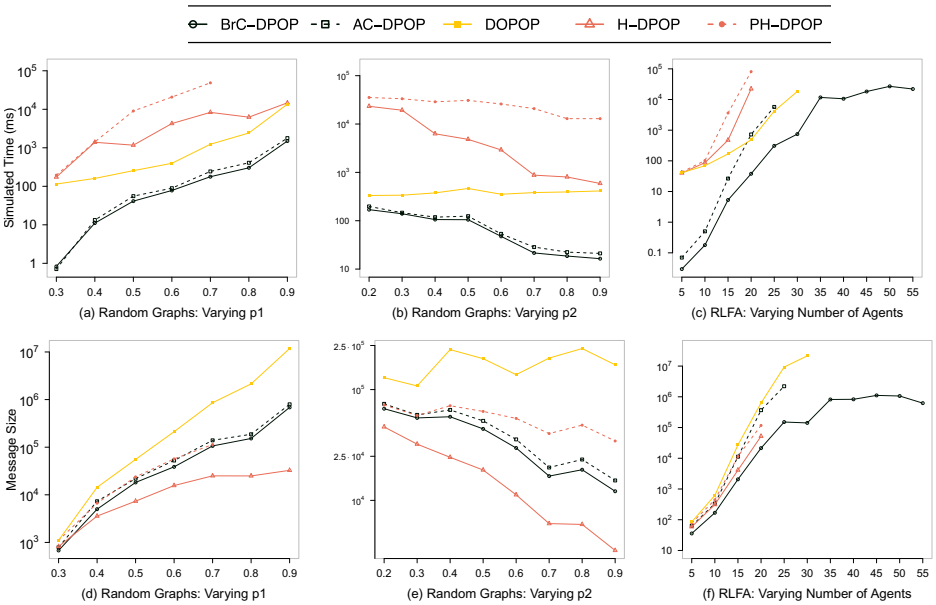


Fig. 3. Runtimes and Message Sizes

common in most DCOP algorithms including BrC-DPOP. Specifically, agents in PH-DPOP can only access their own constraints and, unlike H-DPOP, cannot access their neighboring agents' constraints.

In our experiments,[4] we compare AC-DPOP and BrC-DPOP against DPOP [24], H-DPOP [17], and PH-DPOP. We use a publicly-available implementation of DPOP available in the FRODO framework [18] and an implementation of H-DPOP provided by the authors. We ensure that all algorithms use the same pseudo-tree for fair comparisons. All experiments are performed on an Intel i7 Quadcore 3.4GHz machine with 16GB of RAM with a 300-second timeout. If an algorithm fails to solve a problem, it is due to either memory limitations or timeout. We conduct our experiments on random graphs [6], where we systematically vary the constraint density p_1 and constraint tightness p_2,[5] and distributed Radio Link Frequency Assignment (RLFA) problems [5], where we vary the number of agents $|\mathcal{A}|$ in the problem. We generated 50 instances for each experimental setting, and we report the average runtime, measured using the simulated runtime metric [28], and the average total message size, measured in the number of utility values in the UTIL tables. For the distributed RLFA problems, we also report the percentage of satisfiable instances solved to show the scalability of the algorithms.

Random Graphs: In our experiments, we set $|\mathcal{A}| = 10$, $|\mathcal{X}| = 10$, $|D_i| = 8$ for all variables. We vary p_1 (while setting $p_2 = 0.6$) and vary the p_2 (while setting $p_1 = 0.6$). We did not bound the tree-width, which is determined based on the underlying graph

[4] Available at `http://www.cs.nmsu.edu/klap/brc-dpop_cp14/`

[5] p_1 and p_2 are defined as the ratio between the number of binary constraints in the problem and the maximum possible number of binary constraints in the problem and the ratio between the number of hard constraints and the number of constraints in the problem, respectively.

and randomly generated. We used hard constraints that are either the "less than" or "different" constraints. We also assign a unary constraint to each variable that gives it a utility corresponding to each its value assignments.

Figures 3(a-b) show the runtimes of the algorithms and Figures 3(d-e) show the message sizes. We omit results of an algorithm for a specific parameter if it fails to solve 50% of the satisfiable instances for that parameter. We make the following observations:

- On message sizes, BrC-DPOP uses smaller messages than AC-DPOP because BrC-DPOP prunes more values due to its BrC propagation enforcement. H-DPOP uses smaller messages than BrC-DPOP and AC-DPOP because agents in H-DPOP utilize more information about the neighbors' constraints to prune values. In contrast, agents in BrC-DPOP and AC-DPOP only utilize information on their own constraints to prune values. BrC-DPOP and AC-DPOP use smaller messages than PH-DPOP at large p_2 values, highlighting the strength of the AC and BrC propagation phases compared to the pruning techniques in PH-DPOP. Finally, they all use smaller messages than DPOP because they all prune values while DPOP does not.
- On runtimes, BrC-DPOP is slightly faster than AC-DPOP because BrC-DPOP prunes more values than AC-DPOP. Additionally, these results also indicate that the overhead of the BrC propagation phase is relatively small. BrC-DPOP and AC-DPOP are faster than DPOP because they do not need to perform operations on the pruned values. This also indicates that the overhead of the AC propagation phase is small. In our experiments, the number of *AC* messages exchanged during the AC propagation phase never exceeds $3|\mathcal{F}|$ and is, on average, as small as $|\mathcal{F}|$. DPOP is faster than H-DPOP and PH-DPOP because they maintain and perform operations on CDDs, which are computationally very expensive. In contrast, BrC-DPOP maintains and performs operations on matrices, which are more computationally efficient.

Distributed RLFA Problem: In these problems, we are given a set of links $\{L_1, \ldots, L_r\}$, each consisting of a transmitter and a receiver. Each link must be assigned a frequency from a given set F. At the same time the total interference at the receivers must be reduced below an acceptable level using as few frequencies as possible. In our model, each transmitter corresponds to an agent (and a variable). The domain of each variable consists of the frequencies that can be assigned to the corresponding transmitter. The interference between transmitters are modeled as constraints of the form $|x_i - x_j| > s$, where x_i and x_j are variables and $s \geq 0$ is a randomly generated required frequency separation. We also assign a utility value to each frequency taken by each agent, represented as a unary soft constraint, which represents a preference for an agent to transmit at a given frequency.

For generating the constraint graphs, we vary $|\mathcal{A}|$ and fix the other parameters: $|D_i| = 6, p_2 = 0.55, s \in \{2, 3\}$. We also set the maximum number of neighbors for each agent to 3 in order to generate more satisfiable instances. Figure 3(c) shows the runtimes and Figure 3(f) shows the message sizes. We omit results of an algorithm for a specific parameter if it fails to solve 50% of the satisfiable instances for that parameter.

We observe trends that are similar to those in the earlier random graphs except that the message size of H-DPOP is slightly larger than of those of BrC-DPOP. Therefore, as we have described in Section 5, it is possible for H-DPOP to prune fewer values despite using more information. Additionally, both H-DPOP and PH-DPOP can only solve

Table 2. Percentage of Satisfiable Instances Solved

$\|\mathcal{A}\|$	5	10	15	20	25	30	35	40	45	50	55
BrC-DPOP	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.97**	**0.52**	**0.78**	**0.73**	**0.70**	**0.51**
AC-DPOP	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.39	0.11	0.30	0.15	0.15	0.19
H-DPOP	**1.00**	**1.00**	**1.00**	**1.00**	0.46	0.12	0.00	0.00	0.00	0.00	0.00
PH-DPOP	**1.00**	**1.00**	**1.00**	**1.00**	0.21	0.09	0.00	0.00	0.00	0.00	0.00
DPOP	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.67	0.23	0.35	0.23	0.29	0.19

small problems and failed to solve some problems that DPOP successfully solved. Table 2 tabulates the percentage of satisfiable problem instances solved by each algorithm (the largest percentage in each parameter setting is shown in bold), where it is clear that BrC-DPOP is more scalable than all its counterparts.

7 Conclusions and Future Work

To the best of our knowledge, H-DPOP is the only existing DCOP algorithm that propagates exclusively hard constraints. Unfortunately, it suffers from high computational requirements as well as its overly strong assumption on the knowledge of each agent. In this paper, we alleviate these limitations by introducing the concept of branch consistency as well as the BrC-DPOP algorithm, a DPOP extension that enforces arc consistency and branch consistency. We experimentally show that BrC-DPOP can prune as much as a version of H-DPOP that limits its knowledge to the same amount as BrC-DPOP in a much smaller amount of time. We also show that it can scale to larger problems than DPOP and H-DPOP. Therefore, these results confirm the strengths of this approach, leading to enhanced efficiency and scalability.

For future work, we plan to extend BrC-DPOP to handle higher arity constraints, which can be done by substituting the VRM structures with either consistency graphs or higher dimension VRMs. We suspect that there will be a tradeoff between runtime and memory requirement between the two approaches, where using higher dimension VRMs is faster but uses more memory. We also plan to extend BrC-DPOP to memory-bounded versions similar to MB-DPOP [27] in order to scale to even larger problems. Finally, we plan to explore propagation of soft constraints similar to the versions of BnB-ADOPT with soft AC enforcement [1,13,11].

Acknowledgment. We would like to thank Akshat Kumar for providing us his implementation of H-DPOP. This research is partially supported by the National Science Foundation under grant number HRD-1345232. 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.

References

1. Bessiere, C., Gutierrez, P., Meseguer, P.: Including Soft Global Constraints in DCOPs. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 175–190. Springer, Heidelberg (2012)
2. Bessiere, C., Regin, J.: Refining the Basic Constraint Propagation Algorithm. In: Proc. of IJCAI, pp. 309–315 (2001)
3. Brito, I., Meseguer, P.: Improving DPOP with function filtering. In: Proc. of AAMAS, pp. 141–158 (2010)
4. Burke, D., Brown, K.: Efficiently Handling Complex Local Problems in Distributed Constraint Optimisation. In: Proc. of ECAI, pp. 701–702 (2006)
5. Cabon, B., De Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio Link Frequency Assignment. Constraints 4(1), 79–89 (1999)
6. Erdös, P., Rényi, A.: On Random Graphs I. Publicationes Mathematicae Debrecen 6, 290 (1959)
7. Ezzahir, R., Bessiere, C., Belaissaoui, M., Bouyakhf, E.: DisChoco: A Platform for Distributed Constraint Programming. In: Proc. of the Distributed Constraint Reasoning Workshop, pp. 16–27 (2007)
8. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. In: Proc. of AAMAS, pp. 639–646 (2008)
9. Gershman, A., Meisels, A., Zivan, R.: Asynchronous Forward-Bounding for Distributed COPs. Journal of Artificial Intelligence Research 34, 61–88 (2009)
10. Greenstadt, R., Grosz, B., Smith, M.: SSDPOP: Improving the Privacy of DCOP with Secret Sharing. In: Proc. of AAMAS, pp. 1098–1100 (2007)
11. Gutierrez, P., Lee, J.H.M., Lei, K.M., Mak, T.W.K., Meseguer, P.: Maintaining Soft Arc Consistencies in BnB-ADOPT$^+$ during Search. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 365–380. Springer, Heidelberg (2013)
12. Gutierrez, P., Meseguer, P.: Saving redundant messages in BnB-ADOPT. In: Proc. of AAAI, pp. 1259–1260 (2010)
13. Gutierrez, P., Meseguer, P.: Improving BnB-ADOPT$^+$-AC. In: Proc. of AAMAS, pp. 273–280 (2012)
14. Gutierrez, P., Meseguer, P., Yeoh, W.: Generalizing ADOPT and BnB-ADOPT. In: Proc. of IJCAI, pp. 554–559 (2011)
15. Hamadi, Y., Bessière, C., Quinqueton, J.: Distributed Intelligent Backtracking. In: Proc. of ECAI, pp. 219–223 (1998)
16. Kumar, A., Faltings, B., Petcu, A.: Distributed Constraint Optimization with Structured Resource Constraints. In: Proc. of AAMAS, pp. 923–930 (2009)
17. Kumar, A., Petcu, A., Faltings, B.: H-DPOP: Using Hard Constraints for Search Space Pruning in DCOP. In: Proc. of AAAI, pp. 325–330 (2008)
18. Léauté, T., Ottens, B., Szymanek, R.: FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. In: Proc. of the Distributed Constraint Reasoning Workshop, pp. 160–164 (2009)
19. Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., Varakantham, P.: Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Event Scheduling. In: Proc. of AAMAS, pp. 310–317 (2004)
20. 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)
21. Mohr, R., Henderson, T.C.: Arc and Path Consistency Revisited. Artificial Intelligence 28(2), 225–233 (1986)

22. Nguyen, D.T., Yeoh, W., Lau, H.C.: Distributed Gibbs: A Memory-Bounded Sampling-Based DCOP Algorithm. In: Proc. of AAMAS, pp. 167–174 (2013)
23. Ottens, B., Dimitrakakis, C., Faltings, B.: DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. In: Proc. of AAAI, pp. 528–534 (2012)
24. Petcu, A., Faltings, B.: A Scalable Method for Multiagent Constraint Optimization. In: Proc. of IJCAI, pp. 1413–1420 (2005)
25. Petcu, A., Faltings, B.V.: Approximations in Distributed Optimization. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 802–806. Springer, Heidelberg (2005)
26. Petcu, A., Faltings, B.: ODPOP: An algorithm for open/distributed constraint optimization. In: Proc. of AAAI, pp. 703–708 (2006)
27. Petcu, A., Faltings, B.: MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. In: Proc. of IJCAI, pp. 1452–1457 (2007)
28. Sultanik, E., Lass, R., Regli, W.: DCOPolis: a Framework for Simulating and Deploying Distributed Constraint Reasoning Algorithms. In: Proc. of the Distributed Constraint Reasoning Workshop (2007)
29. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. Journal of Artificial Intelligence Research 38, 85–133 (2010)
30. Yeoh, W., Yokoo, M.: Distributed Problem Solving. AI Magazine 33(3), 53–65 (2012)
31. Yokoo, M. (ed.): Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems. Springer (2001)
32. 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)
33. Zivan, R., Glinton, R., Sycara, K.: Distributed Constraint Optimization for Large Teams of Mobile Sensing Agents. In: Proc. of IAT, pp. 347–354 (2009)
34. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. Artificial Intelligence 212, 1–26 (2014)