

RMB-DPOP: Refining MB-DPOP by Reducing Redundant Inferences

Ziyu Chen
College of Computer Science,
Chongqing University
Chongqing, China
chenziyu@cqu.edu.cn

Wenxin Zhang
College of Computer Science,
Chongqing University
Chongqing, China
wenxinzhang18@163.com

Yanchen Deng*
College of Computer Science,
Chongqing University
Chongqing, China
dyc941126@126.com

Dingding Chen
College of Computer Science,
Chongqing University
Chongqing, China
dingding@cqu.edu.cn

Qing Li
College of Electrical Engineering,
Chongqing University
Chongqing, China
qiangli.ac@gmail.com

ABSTRACT

MB-DPOP is an important complete algorithm for solving Distributed Constraint Optimization Problems (DCOPs) by exploiting a cycle-cut idea to implement memory-bounded inference. However, each cluster root in the algorithm is responsible for enumerating all the instantiations of its cycle-cut nodes, which would cause redundant inferences when its branches do not have the same cycle-cut nodes. Additionally, a large number of cycle-cut nodes and the iterative nature of MB-DPOP further exacerbate the pathology. As a result, MB-DPOP could suffer from huge coordination overheads and cannot scale up well. Therefore, we present RMB-DPOP which incorporates several novel mechanisms to reduce redundant inferences and improve the scalability of MB-DPOP. First, using the independence among the cycle-cut nodes in different branches, we distribute the enumeration of instantiations into different branches whereby the number of nonconcurrent instantiations reduces significantly and each branch can perform memory bounded inference asynchronously. Then, taking the topology into the consideration, we propose an iterative allocation mechanism to choose the cycle-cut nodes that cover a maximum of active nodes in a cluster and break ties according to their relative positions in a pseudo-tree. Finally, a caching mechanism is proposed to further reduce unnecessary inferences when the historical results are compatible with the current instantiations. We theoretically show that with the same number of cycle-cut nodes RMB-DPOP requires as many messages as MB-DPOP in the worst case and the experimental results show our superiorities over the state-of-the-art.

KEYWORDS

DCOP, complete algorithms, memory-bounded inference

ACM Reference Format:

Ziyu Chen, Wenxin Zhang, Yanchen Deng, Dingding Chen, and Qing Li. 2020. RMB-DPOP: Refining MB-DPOP by Reducing Redundant Inferences. In *Proc. of the 19th International Conference on Autonomous Agents and*

*Corresponding author

Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020, IFAAMAS, 9 pages.

1 INTRODUCTION

Distributed constraint optimization problems (DCOPs) are a fundamental framework for coordinated and cooperative multi-agent systems. DCOPs have been successfully applied to model many real-world problems including sensor networks [30], meeting scheduling [26], smart grid [9], etc.

Incomplete algorithms for DCOPs [4, 8, 17, 19, 20, 30] aim to find a good solution in an acceptable overhead, while complete algorithms focus on finding the optimal one by employing either search or inference to systematically explore the entire solution space. SBB [14], AFB [11], PT-FB [16], ADOPT [18] and its variants [12, 13, 27–29] are typical search-based algorithms that employ distributed backtrack search to exhaust the search space. However, these algorithms incur a prohibitively large number of messages and can only solve the problems with a handful of variables.

On the other hand, inference-based complete algorithms like DPOP [22] perform dynamic-programming on a pseudo-tree and only require a linear number of messages. However, the memory consumption in DPOP is exponential to the induced width [6], which makes it not applicable for the memory-limited scenarios where the optimal solution is desired [7, 15]. Therefore, a number of algorithms [2, 21, 23–25] were proposed to trade either solution quality or message number for smaller memory consumption. Among these algorithms, MB-DPOP [25] iteratively performs memory-bounded utility propagation to guarantee the optimality. Specifically, given the dimension limit k , the algorithm first identifies high-width areas (clusters) and cycle-cut nodes [6] to make the maximal dimension of the utility tables propagated within clusters no greater than k . For each cluster, the cluster root is responsible for iteratively enumerating all the instantiations of its cycle-cut nodes, and nodes in the cluster perform memory-bounded inferences by conditioning utility tables on these instantiations. Once instantiations are exhausted, the cluster root propagates the resulted utility table to its parent.

However, a key limitation in MB-DPOP is the inability of exploiting the structure of a problem. As a result, MB-DPOP suffers from a

severe redundancy in memory-bounded inference. First, since each cluster root enumerates for all its cycle-cut nodes without considering the independence of cycle-cut nodes in different branches, each branch in a cluster would have to perform redundant inferences when there are cycle-cut nodes which have nothing to do with the branch. Also, these nonconcurrent instantiations severely degenerate the parallelism among branches. Second, agents in a cluster use heuristics to determine cycle-cut nodes locally, which would result in a large number of cycle-cut nodes. Finally, MB-DPOP ignores the validity of the inference results and a branch has to perform inference even if the previous results are compatible with the current instantiations.

In this paper, we aim to improve the scalability of MB-DPOP by exploiting the structure of a problem. More specifically, our contributions are listed as follows.

- By using the independence among the cycle-cut nodes in different branches, we propose a distributed enumeration mechanism where a cluster root only enumerates for cycle-cut nodes in its separators and these instantiations are augmented with branch-specific cycle-cut nodes dynamically along the propagation. Accordingly, each branch can perform memory-bounded inferences asynchronously and the number of nonconcurrent instantiations can be reduced significantly.
- We propose an iterative selection mechanism to determine cycle-cut nodes by taking both their effectiveness and positions in a pseudo tree into consideration. Concretely, rather than choosing highest/lowest separators as cycle-cut nodes, we tend to choose the nodes that cover a maximum of active nodes in a cluster and break ties according to their relative positions. Moreover, we propose a caching mechanism to exploit the historical inference results which are compatible with the current instantiations to further avoid unnecessary utility propagation.
- We theoretically show that the message number of our algorithm is no more than the one in MB-DPOP. Our empirical evaluation confirms the superiority of our algorithm over the state-of-the-art on various benchmarks.

The rest of this paper is organized as follows. Section 2 gives the background including DCOPs, pseudo tree, DPOP and MB-DPOP. In Section 3, we give the motivation and describe the details of our proposed algorithm. Finally, the experiments are shown in Section 4 and Section 5 concludes this paper.

2 BACKGROUND

In this section, we introduce the preliminaries including DCOPs, pseudo tree, DPOP and MB-DPOP.

2.1 Distributed Constraint Optimization Problems

A distributed constraint optimization problem [22] can be represented by a tuple $\langle A, X, D, F \rangle$ where

- $A = \{a_1, \dots, a_n\}$ is a set of agents
- $X = \{x_1, \dots, x_m\}$ is a set of variables

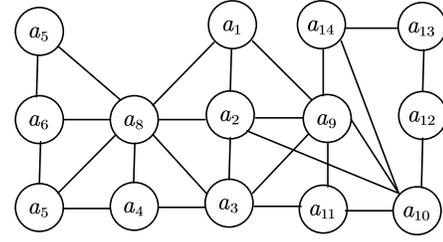


Figure 1: The constraint graph of a DCOP

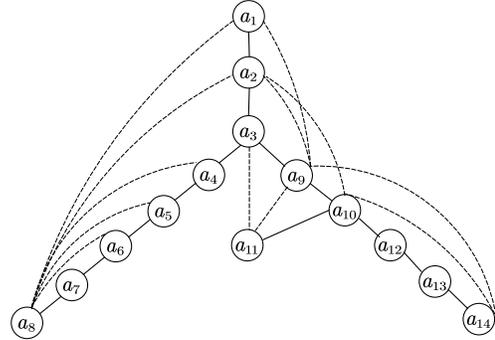


Figure 2: A pseudo tree derived from Figure 1

- $D = \{D_1, \dots, D_m\}$ is a set of domains that are finite and discrete, each variable x_i taking a value assignment from D_i
- $F = \{f_1, \dots, f_q\}$ is a set of constraint functions, each function $f_i : D_{i_1} \times \dots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0}$ denoting the non-negative cost for each assignment combination of x_{i_1}, \dots, x_{i_k} .

For the sake of simplicity, we assume that each agent controls a variable and all constraint functions are binary (i.e., $f_{ij} : D_i \times D_j \rightarrow \mathbb{R}_{\geq 0}$). Here, the term "agent" and "variable" can be used interchangeably. A solution to a DCOP is an assignment including all variables that makes the minimum cost. That is

$$X^* = \arg \min_{d_i \in D_i, d_j \in D_j} \sum_{f_{ij} \in F} f_{ij}(x_i = d_i, x_j = d_j)$$

A DCOP can be visualized by a constraint graph presented as Fig. 1, where the nodes represent the agents and the edges represent the constraints, respectively.

2.2 Pseudo Tree

A pseudo tree [10] is a partial ordering among agents and can be generated by depth-first search (DFS) traversal to a constraint graph, where different branches are independent from each other. Given a constraint graph and its spanning tree, the edges in the spanning tree are tree edges and the other edges are pseudo edges (i.e., non-tree edges). According to the relative positions in a pseudo tree, the neighbors of an agent a_i connected by tree edges are categorized into parent $P(a_i)$ and children $C(a_i)$, while the ones connected by pseudo edges are denoted as pseudo parents $PP(a_i)$ and pseudo children $PC(a_i)$. For its parent and pseudo parents, we denote them as $AP(a_i) = \{P(a_i)\} \cup PP(a_i)$. We also denote its descendants as $Desc(a_i)$. Finally, the separators $Sep(a_i)$ [23]

of a_i refer to the ancestors which are constrained with a_i or its descendants. Fig. 2 gives a pseudo tree derived from the constraint graph in Fig. 1, where tree edges and pseudo edges are denoted by solid and dotted lines, respectively.

2.3 DPOP

DPOP [22] is an inference-based complete algorithm for DCOPs, which implements the bucket elimination in a distributed manner [5]. It performs two phases of propagation on a pseudo tree via tree edges: a UTIL propagation phase to eliminate dimensions from the bottom up, and a VALUE propagation phase to assign the optimal value for each variable vice versa along the pseudo tree. More specifically, in a UTIL propagation phase, an agent a_i collects the utility tables from its children and joins them with the local utility table, then computes the optimal utilities for all possible assignments of $Sep(a_i)$ to eliminate its dimension from the joint utility table. Then, it sends the projected utility table to its parent. In a VALUE propagation phase, once a_i receives the assignments from its parent, it plugs them into the joint utility table obtained in the UTIL propagation phase to get the optimal assignment, and sends the joint assignment to its children. Although DPOP only requires a linear number of messages to solve a DCOP, its space complexity is exponential in the induced width of the pseudo tree.

2.4 MB-DPOP

MB-DPOP [25] attempts to improve the scalability of DPOP by trading the message number for smaller memory consumption. Given the dimension limit k , MB-DPOP starts with a labeling phase to identify the areas with the induced width [6] higher than k (i.e., clusters) and the corresponding cycle-cut (CC) nodes. Each cluster is bounded at the top by the lowest node in the tree that has separators of size k or less, and such node is called the cluster root (CR) node. For each clusters, CC nodes are determined such that the cluster has the width no greater than k once they are removed.

In more detail, the CC nodes are selected and then aggregated in a bottom-up fashion. That is, given the lists of CC nodes selected by its children, a_i first determines whether its width exceeds k if $|Sep(a_i)| > k$. If it is the case, a_i needs to choose additional CC nodes to enforce the memory limit k by a heuristic function. Then, a_i propagates all the CC nodes $CCList_i$ to its parent $P(a_i)$. Otherwise, if $|Sep(a_i)| \leq k$ and the lists received from children are all empty, a_i labels self as a normal node and propagates the utility as in DPOP.

During the UTIL propagation phase normal nodes (i.e., the nodes whose width is no greater than k) perform canonical utility propagation while the other nodes in each cluster perform memory-bounded inferences. Specifically, each cluster root (CR) enumerates instantiations for its CC nodes and propagates them iteratively to the other nodes in the cluster, and these nodes perform memory-bounded inferences by conditioning the utility tables on the received instantiation. The cluster root eliminates its dimension and propagates the utility table to its parent after exhausting all the combinations. Finally, a VALUE propagation phase starts. Different from DPOP which only requires a round of value propagation, MB-DPOP requires additional utility propagation to re-drive the utilities corresponding to the assignments of CC nodes to get the

optimal values, since non-CC nodes in a cluster only cache the utility table for the latest instantiation of CC nodes.

3 PROPOSED METHOD

In this section, we present our proposed RMB-DPOP. We begin with a motivation, and then present the details and the theoretical claim of our algorithm, respectively.

3.1 Motivation

As stated earlier, MB-DPOP suffers from plenty of redundancies in memory-bounded inference due to the inability of exploiting a problem structure in both instantiation enumeration and the selection of CC nodes. Consider the problem in Fig. 2, where a_3 is the only cluster root with the dimension limit $k = 2$. Since each agent in the cluster selects its CC nodes only with the local knowledge, MB-DPOP would select a large number of CC nodes and significantly increase the number of instantiations. In fact, if we choose CC nodes with the highest level, the CC nodes of a_3 are $CCList_3 = \{a_1, a_2, a_3, a_4, a_5, a_9\}$. It would be worse when using the lowest heuristic which results in 9 CC nodes in this case. Alternatively, instead of choosing both a_3 and a_9 , we could only choose a_9 and still guarantee the memory budget. Besides, the cluster root a_3 has to enumerate all instantiations of $CCList_3$, which results in a large number of nonconcurrent instantiations and redundant inferences. In fact, we could exploit the independence between branch a_4 and branch a_9 by generating instantiations that only contains the common CC nodes (i.e., a_1, a_2 and a_3). In this way, branch a_4 and branch a_9 can operate asynchronously and the number of non-concurrent instantiations is significantly reduced. In addition, all the bounded inference results are disposable in MB-DPOP, which also leads to redundant inferences. In fact, some inference results received from children in the previous iterations are compatible with the current instantiation, since each branch performs memory-bounded inference by conditioning only on a subset of all cycle-cut nodes of a cluster. Thus, it is unnecessary to perform a memory-bounded inference when the assignments of corresponding CC nodes do not change.

Therefore, to take the structure of a problem into consideration, we propose a novel algorithm named RMB-DPOP which incorporates a distributed enumeration mechanism to reduce the nonconcurrent instantiations, an iterative selection mechanism to reduce the number of CC nodes and a caching mechanism to avoid unnecessary inferences. Algorithm 1¹ presents the sketch of RMB-DPOP.

3.2 Distributed Enumeration Mechanism

Distributed enumeration mechanism (DEM) is adopted in each cluster to perform asynchronous memory-bounded inference by factorizing the instantiations. More specifically, since each branch in a pseudo tree is independent, each CC node inside a cluster is only related to a subproblem. Hence, instead of enumerating all the instantiations of CC nodes by a cluster root, we only generate instantiations for the CC nodes in the separators of the cluster

¹We omit the details of the value propagation phase due to its similarity to the one in MB-DPOP. The source code is available in <https://github.com/czy920/RMB-DPOP>.

Algorithm 1: RMB-DPOP for each agent a_i

```

When Initialization:
1 | start Labeling phase
When Labeling phase finished:
2 | if  $TYPE(a_i) = CR$  then
3 |    $curIns_i \leftarrow$  the first instantiation of  $(Sep(a_i) \cup \{a_i\}) \cap CCList_i$ 
4 |    $Local\_util_i \leftarrow$  the join of local utilities w.r.t.  $AP(a_i)$ 
5 |   PropagateInstantiation()
6 | else if  $TYPE(a_i) = NORMAL \wedge isLeafAgent()$  then
7 |    $u_i \leftarrow$  the join of local utilities w.r.t.  $AP(a_i)$ 
8 |    $u_i \leftarrow \min_{x_i} u_i$ 
9 |   send  $NORMAL\_UTIL(u_i)$  to  $P(a_i)$ 
When received  $NORMAL\_UTIL(u_c)$  from  $a_c \in C(a_i)$ :
10 |  $Normal\_utils_i \leftarrow Normal\_utils_i \cup \{u_c\}$ 
11 | if received utilities from all children then
12 |   if  $isRootAgent()$  then
13 |     start Value propagation phase
14 |   else if  $TYPE(a_i) = NORMAL$  then
15 |      $u_i \leftarrow \bigotimes_{u_c \in Normal\_utils_i} u_c$ 
16 |     update  $u_i$  with local utilities w.r.t.  $AP(a_i)$ 
17 |      $u_i \leftarrow \min_{x_i} u_i$ 
18 |     send  $NORMAL\_UTIL(u_i)$  to  $P(a_i)$ 
When received  $INSTANTIATION(Ins)$  from  $P(a_i)$ :
19 |  $Bounded\_utils_i \leftarrow \emptyset, Ins_i \leftarrow Ins_{CCList_i}$ 
20 |  $Local\_util_i \leftarrow$  local utilities w.r.t.  $AP(a_i)$  conditioned on  $Ins_i$ 
21 | if  $TYPE(a_i) \neq CL$  then
22 |   if  $TYPE(a_i) = CC$  then
23 |      $curValue_i \leftarrow$  the first element in  $D_i$ 
24 |     PropagateInstantiation()
25 |   else if received utilities from all children then
26 |     send bounded utilities conditioned on  $Ins_i$ 
27 |     via  $BOUNDED\_UTIL(bu_i)$  to  $P(a_i)$ 
When received  $BOUNDED\_UTIL(bu_c)$  from  $a_c \in C(a_i)$ :
28 |  $Bounded\_utils_i \leftarrow Bounded\_utils_i \cup \{bu_c\}$ 
29 | if received utilities from all children then
30 |   if  $TYPE(a_i) = CC$  then
31 |      $bu_i \leftarrow Update(bu_i, Bounded\_utils_i,$ 
32 |        $Local\_util_i, Normal\_utils_i)$ 
33 |     if  $curValue_i.next() \neq null$  then
34 |        $curValue_i \leftarrow curValue_i.next()$ 
35 |       PropagateInstantiation()
36 |     else
37 |       send  $BOUNDED\_UTIL(bu_i)$  to  $P(a_i)$ 
38 |   if  $TYPE(a_i) = CR$  then
39 |      $u_i \leftarrow Update(u_i, Bounded\_utils_i,$ 
40 |        $Local\_util_i, Normal\_utils_i)$ 
41 |     if  $curIns_i.next() \neq null$  then
42 |        $curIns_i \leftarrow curIns_i.next()$ 
43 |       PropagateInstantiation()
44 |     else
45 |       send  $NORMAL\_UTIL(u_i)$  to  $P(a_i)$ 
46 |   else
47 |     join all utilities conditioned on  $Ins_i$  into one utility table  $bu_i$ 
48 |     send  $BOUNDED\_UTIL(bu_i)$  to  $P(a_i)$ 
Function  $PropagateInstantiation()$ :
49 | if  $TYPE(a_i) = CC$  then
50 |    $Ins_i \leftarrow Ins_i \cup (x_i, curValue_i)$ 
51 | else if  $TYPE(a_i) = CR$  then
52 |    $Ins_i \leftarrow curIns_i$ 
53 | send  $INSTANTIATION(Ins_i)$  to  $\forall a_c \in C(a_i)$  s.t.  $CCList_c \neq \emptyset$ 

```

root and dynamically augment these instantiations with branch-specific CC nodes. In the following, we present the details of the mechanism.

When the Labeling phase finishes, a CR node a_j starts the iterative memory-bounded UTIL propagation by instantiating the nodes in $(Sep(a_j) \cup \{a_j\}) \cap CCList_j$ (line 2-5), where the $CCList_j$ is a list of the CC nodes corresponding to the branch of a_j . When a

CC node a_i receives an instantiation Ins_i from its parent, it augments Ins_i by the first assignment from its domain and propagates the extended instantiation to its children in the cluster (line 22-24, 46-50). Once a_i receives all the utilities from its children, it updates the cache and replaces self assignment with the next value in its domain, and then propagates the new instantiation (line 29-33). Until getting a complete bounded inference result corresponding to Ins_i by a traversal of its domain, a_i sends the result to its parent via a $BOUNDED_UTIL$ message (line 35).

Next, we theoretically show its superiority over MB-DPOP in terms of the message number. Let us first introduce two notations. For a cluster root a_j , we denote $CCList_j^{out} = (Sep(a_j) \cup \{a_j\}) \cap CCList_j$ as the set of CC nodes enumerated by a_j , and the remaining CC nodes as $CCList_j^{in} = CCList_j \setminus CCList_j^{out}$.

LEMMA 3.1. *For an agent a_i in a cluster where a_j is the CR node, the number of instantiations it receives is exponential in the size of $(CCList_i \cap Sep(a_i)) \cup CCList_j^{out}$.*

PROOF. According to line 2-5, the nonconcurrent instantiations sent from a_j is exponential in $|CCList_j^{out}|$. Besides, each nonconcurrent instantiation is augmented by the CC nodes along the path from a_j to a_i (line 23, 32). Therefore, the number of instantiations a_i receives is exponential in $|(CCList_i \cap Sep(a_i)) \cup CCList_j^{out}|$. \square

THEOREM 3.2. *Given the same CC lists of each node in each cluster, the maximal message number of RMB-DPOP is no more than the one in MB-DPOP.*

PROOF. It is enough to show the theorem by analyzing the total number of instantiations received by each agent a_i in a cluster, since a_i must respond with a bounded utility table to its parent after receiving an instantiation. Without loss of generality, we assume that each variable has a domain with the same size d . In MB-DPOP each agent in a cluster will receive $d^{|CCList_j|}$ instantiations. Whereas from Lemma 1, we have the number of instantiations sent to a_i as $d^{|(CCList_i \cap Sep(a_i)) \cup CCList_j^{out}|}$, and

$$\begin{aligned}
 |CCList_j| &= |CCList_j^{out} \cup CCList_j^{in}| \\
 &= |CCList_j^{out}| + |CCList_j^{in}| \\
 &\geq |CCList_j^{out}| + |CCList_i \cap CCList_j^{in} \cap Sep(a_i)| \\
 &= |CCList_j^{out} \cup (CCList_i \cap CCList_j^{in} \cap Sep(a_i))| \\
 &= |(CCList_i \cap Sep(a_i)) \cup CCList_j^{out}|
 \end{aligned}$$

Consequently, RMB-DPOP propagates a smaller number of instantiations than MB-DPOP. And only when the cluster does not have the CC nodes inside the cluster (i.e., $CCList_j^{in} = \emptyset$), the instantiations for each agent in RMB-DPOP are equivalent to those in MB-DPOP. Thus, the theorem holds. \square

3.3 Iterative Selection Mechanism

Instead of selecting CC nodes based on the local knowledge in MB-DPOP which would result in a large number of CC nodes, we choose CC nodes by taking their effectiveness and their relative positions into consideration through an iterative selection mechanism (ISM). Specifically, in a cluster we measure the effectiveness of a node by the number of active nodes it covers. Here, an active node is

Algorithm 2: Labeling phase for each agent a_i

```

When Initialization:
1  determine the node type via a bottom-up propagation
2  if  $TYPE(a_i) = NORMAL$  then
3    terminate the Labeling phase
4    return
5   $Eff_i \leftarrow \emptyset, dr_c^* \leftarrow 0, CCList_i \leftarrow \emptyset$ 
6  if  $TYPE(a_i) = CL$  then
7    propagateSepInfo()
When received  $SEP\_INFO(Effc, dr_c^*, cc)$  from  $a_c$ :
8  if  $cc \neq null$  then
9     $CCList_i \leftarrow CCList_i \cup \{cc\}$ 
10    $curCC \leftarrow cc$ 
11   if  $a_i = cc$  then
12     mark  $a_i$  as a CC node
13    $Eff_i \leftarrow merge(Effc, Eff_i)$ 
14    $dr_i^* \leftarrow \max(dr_i^*, dr_c^*)$ 
15   if receive all  $SEP\_INFO$  from the children in the cluster then
16     if  $TYPE(a_i) = CR$  then
17       if  $Eff_i = \emptyset$  then
18         terminate the Labeling phase
19         send TERMINATE to the children in the cluster
20       else
21          $cc \leftarrow \arg \max_{a_k} Eff_i(a_k)$ 
22         send ALLOCATION(cc) to the children in the cluster
23       else
24         propagateSepInfo()
When received  $ALLOCATION(cc)$  from  $P(a_i)$ :
25   if  $cc \in Sep(a_i)$  then
26      $CCList_i \leftarrow CCList_i \cup \{cc\}, curCC \leftarrow cc$ 
27   else
28      $curCC \leftarrow null$ 
29    $Eff_i \leftarrow \emptyset, dr_c^* \leftarrow 0$ 
30   if  $TYPE(a_i) \neq CL$  then
31     send ALLOCATION(cc) to the children in the cluster
32   else
33     propagateSepInfo()
When received  $TERMINATE$  from  $P(a_i)$ :
34   terminate the Labeling phase
35   if  $TYPE(a_i) \neq CL$  then
36     send TERMINATE to the children in the cluster
Function propagateSepInfo():
37    $R\_Sep_i \leftarrow Sep_i \setminus CCList_i$ 
38   if  $|R\_Sep_i| > k$  then
39     foreach  $a_j \in R\_Sep_i$  do
40       if  $a_j \in Eff_i$  then
41         increase  $Eff_i(a_j)$  by 1
42       else
43          $Eff_i \leftarrow Eff_i \cup \{(a_j, 1)\}$ 
44   if  $a_i \in Eff_i$  then
45      $EffDesc \leftarrow keys(Eff_i) \cap Desc(a_i)$ 
46     if  $Eff_i(a_i) > dr_i^*$  then
47        $dr_i^* \leftarrow Eff_i(a_i)$ 
48       remove  $\forall a_j \in EffDesc$  from  $Eff_i$ 
49     else
50       remove  $a_i$  from  $Eff_i$ 
51        $a_k \leftarrow \arg \max_{a_k \in EffDesc} Eff_i(a_k)$ 
52       remove  $\forall a_j \in EffDesc \wedge a_j \neq a_k$  from  $Eff_i$ 
53   send  $SEP\_INFO(Eff_i, dr_c^*, curCC)$  to  $P(a_i)$ 

```

the one whose width is still greater than k given the selected CC nodes. Besides, to facilitate DEM, we tend to select nodes in different branches of a cluster. Therefore, we propose to break ties among the nodes with the same effectiveness by their positions in a pseudo tree. Algorithm 2 gives the sketch of Labeling phase.

In more detail, a CC node is selected through two phases of message-passing. In the first phase, the effectiveness of each CC node candidate is aggregated in a bottom-up fashion via SEP_INFO

Table 1: The first round of effectiveness aggregation

step	message	SEP_INFO
1	$a_8 \rightarrow a_7$	$Eff_8 = \{(a_1, 1), (a_2, 1), (a_4, 1), (a_5, 1), (a_6, 1), (a_7, 1)\}$
	$a_{14} \rightarrow a_{13}$	$Eff_{14} = \{(a_9, 1), (a_{10}, 1), (a_{13}, 1)\}$
	$a_{11} \rightarrow a_{10}$	$Eff_{11} = \{(a_3, 1), (a_9, 1), (a_{10}, 1)\}$
2	$a_7 \rightarrow a_6$	$Eff_7 = \{(a_1, 2), (a_2, 2), (a_4, 2), (a_5, 2), (a_6, 2), (a_7, 1)\}$
	$a_{13} \rightarrow a_{12}$	$Eff_{13} = \{(a_9, 2), (a_{10}, 2), (a_{12}, 1), (a_{13}, 1)\}$
3	$a_6 \rightarrow a_5$	$Eff_6 = \{(a_1, 3), (a_2, 3), (a_4, 3), (a_5, 3), (a_6, 2)\}$
	$a_{12} \rightarrow a_{10}$	$Eff_{12} = \{(a_9, 3), (a_{10}, 3), (a_{13}, 1)\}$
4	$a_5 \rightarrow a_4$	$Eff_5 = \{(a_1, 4), (a_2, 4), (a_4, 4), (a_5, 3)\}$
	$a_{10} \rightarrow a_9$	$Eff_{10} = \{(a_2, 1), (a_3, 2), (a_9, 5), (a_{10}, 4)\}$
5	$a_4 \rightarrow a_3$	$Eff_4 = \{(a_1, 5), (a_2, 5), (a_3, 1), (a_4, 4)\}$
	$a_9 \rightarrow a_3$	$Eff_9 = \{(a_1, 1), (a_2, 2), (a_3, 3), (a_9, 5)\}$
6	N/A	$Eff_3 = \{(a_1, 6), (a_2, 7), (a_3, 4), (a_4, 4), (a_9, 5)\}$

messages. Specifically, each agent a_i maintains a data structure Eff_i to record the effectiveness of candidates. When receiving a SEP_INFO message from a child a_c , it updates Eff_i by Eff_c according to

$$Eff_i(a_k) = \begin{cases} Eff_c(a_k), & a_k \notin keys(Eff_i) \\ Eff_i(a_k) + Eff_c(a_k), & otherwise \end{cases}, \forall a_k \in keys(Eff_c)$$

If a_i is an active node (i.e., satisfying line 38), for each CC node candidate $a_k \in R_Sep_i$ it increases the effectiveness $Eff_i(a_k)$ by 1 (line 39-43). Then a_i removes all the CC candidates that have a suboptimal effectiveness in its descendants from Eff_i (line 45-52), since they cannot produce the highest effectiveness. The phase ends when the cluster root a_j receives all the SEP_INFO messages from the children in the cluster.

In the second phase, the cluster root a_j chooses the CC node with the maximal effectiveness (line 21) and propagates it into the cluster via ALLOCATION messages. According to Lemma 3.1 and Theorem 3.2, our algorithm can take the advantage of the CC nodes inside the cluster through the DEM. Therefore, we propose to break ties according to the height of the candidates when choosing a CC node, i.e., we tend to choose the lowest CC node since it is more likely to be inside the cluster. The phase ends after each cluster leaf (CL) starts a new phase of effectiveness propagation (line 33). The Labeling phase terminates when there is no active nodes (i.e., satisfying line 17).

It is worth noting that our selection mechanism only incurs minor messages. Specifically, to determine a CC node in the cluster with CR a_j , agents need to propagate bottom-up SEP_INFO messages and top-down ALLOCATION messages via tree edges, which requires $O(m)$ messages. Here, m is the total number of nodes in the cluster. Thus, the total messages exchanged in the Labeling phase in a cluster is $O(|CCList_j| * m)$ and the overall complexity is $O(N^2)$ where N is the total number of agents.

3.4 Caching Mechanism

The caching mechanism attempts to reduce unnecessary inferences by exploiting the historical results when they are compatible with the current instantiations. To do this, before a_i propagates an instantiation to a child a_c , it projects the instantiation on $CCList_c$ and stores the projected one. When a_i receives a new instantiation,

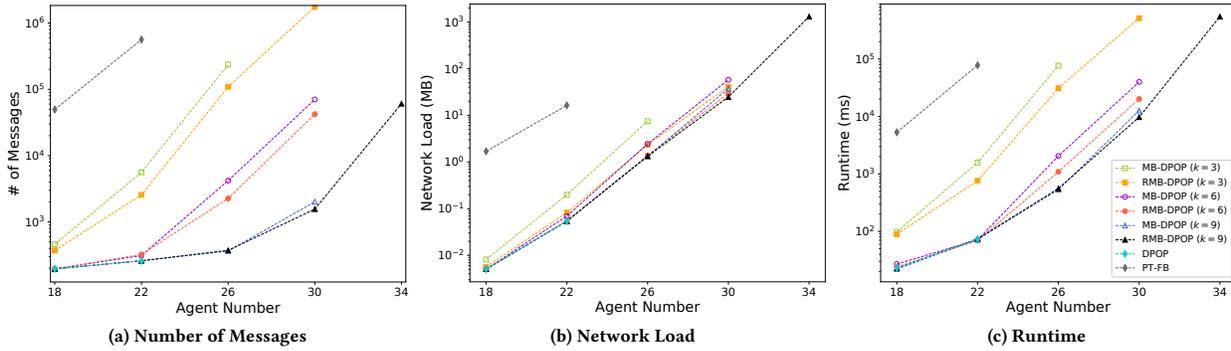


Figure 3: Performance comparison under different agent numbers

Table 2: The CClist of each node in the cluster

a_i	$CClist_i$
a_3	$\{a_2, a_9, a_5, a_6, a_7\}$
a_4	$\{a_2, a_5, a_6, a_7\}$
a_5	$\{a_2, a_5, a_6, a_7\}$
a_6	$\{a_2, a_5, a_6, a_7\}$
a_7	$\{a_2, a_5, a_6, a_7\}$
a_8	$\{a_2, a_5, a_6, a_7\}$
a_9	$\{a_2, a_9\}$
a_{10}	$\{a_2, a_9\}$
a_{11}	$\{a_2, a_9\}$
a_{12}	$\{a_2, a_9\}$
a_{13}	$\{a_2, a_9\}$
a_{14}	$\{a_2, a_9\}$

Table 3: The first round of instantiation propagation

step	message	INSTANTIATION
1	$a_3 \rightarrow a_4$	$\{a_2 = 0\}$
	$a_3 \rightarrow a_9$	$\{a_2 = 0\}$
2	$a_4 \rightarrow a_5$	$\{a_2 = 0\}$
	$a_9 \rightarrow a_{10}$	$\{a_2 = 0, a_9 = 0\}$
3	$a_5 \rightarrow a_6$	$\{a_2 = 0, a_5 = 0\}$
	$a_{10} \rightarrow a_{11}$	$\{a_2 = 0, a_9 = 0\}$
	$a_{10} \rightarrow a_{12}$	$\{a_2 = 0, a_9 = 0\}$
4	$a_6 \rightarrow a_7$	$\{a_2 = 0, a_5 = 0, a_6 = 0\}$
	$a_{12} \rightarrow a_{13}$	$\{a_2 = 0, a_9 = 0\}$
5	$a_7 \rightarrow a_8$	$\{a_2 = 0, a_5 = 0, a_6 = 0, a_7 = 0\}$
	$a_{13} \rightarrow a_{14}$	$\{a_2 = 0, a_9 = 0\}$

for each child it checks whether the instantiation is compatible with the cached one associated with the child. If it is the case, the results cached in the previous iteration is valid and there is no need to perform a memory-bounded inference. Otherwise, the results from the child is no longer valid and a_i propagates the (augmented) instantiation to the child to initiate a new memory-bounded inference.

3.5 Execution Example

For better understanding, we take Fig. 2 as an example to illustrate our algorithm. Assuming the dimension limit $k = 2$, there is only a cluster whose CR node is a_3 . The labeling phase begins with CL nodes a_8 and a_{14} which send SEP_INFO messages to their parents and Table 1 presents the trace of effectiveness aggregation in the first round in a chronological order.

It can be seen that node a_2 has the highest effectiveness and we should choose it as a CC node. Then a top-down phase is initiated to apply a_i into the cluster. These two phases are performed alternatively until all the nodes in the cluster have a width less than k . The final CC nodes for each agent is listed as Table 2.

Then, the DEM begins with a_3 which sends the first instantiation w.r.t. $CClist_3^{out} = \{a_2\}$ to its children a_4 and a_9 . When receiving

an instantiation, a CC node appends its assignment into the instantiation. Table 3 gives the trace of the first round of instantiation propagation.

4 EXPERIMENTAL EVALUATION

In this section, we compare our proposed RMB-DPOP with the state-of-the-art on various benchmarks, and present an ablation study to demonstrate the effectiveness of each mechanism.

4.1 Experimental Configuration

We empirically evaluate RMB-DPOP, PT-FB, DPOP and MB-DPOP on two types of problems, i.e., random DCOPs and scale-free networks [1]. In the first configuration, we consider the random DCOPs with the graph density of 0.2, the domain size of 3 and the agent number varying from 18 to 34. The second configuration is the DCOPs with 20 agents, the graph density of 0.2 and the domain size varying from 3 to 6. In addition, we present the ratio of the problems successfully solved within limited time on the second configuration where the graph density is set to 0.5. In the third configuration, we consider the scale-free networks generated by Barabási-Albert model where we set the agent number to 26, the domain size to 3 and m_0 to 10 and vary m_1 from 2 to 10.

In our experiments, we use the message number and network load (i.e., the size of total information exchanged) to measure the communication overheads. Also, we use wall clock time to measure

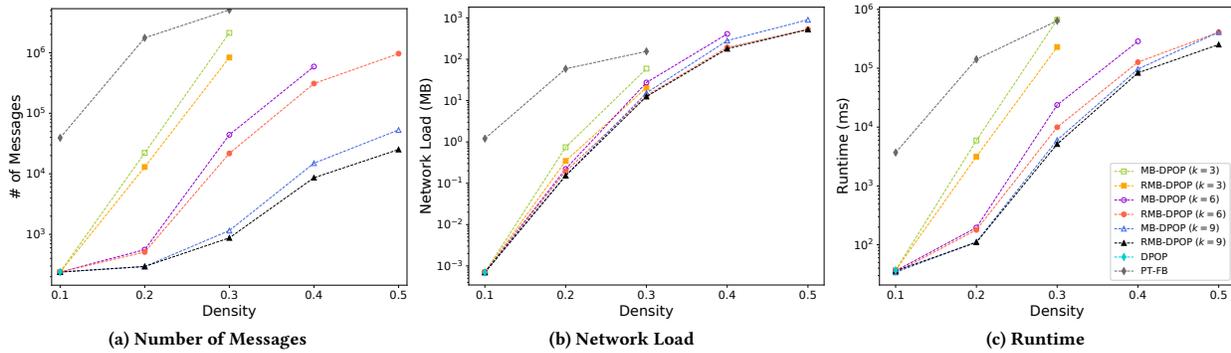


Figure 4: Performance comparison under different graph densities

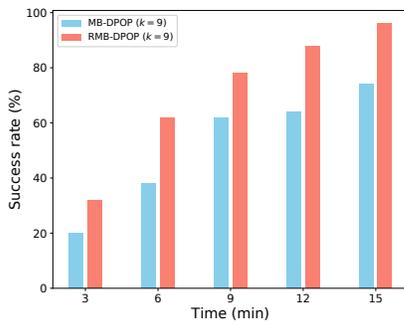


Figure 5: Success rate within limited time

the runtime. For each experiment, we generate 50 random instances and report the medium over all the instances. The experiments are conducted on an i7-7820x workstation with 32GB of memory and we set the timeout to 30 minutes for each algorithm. To demonstrate the effects of different k , we benchmark RMB-DPOP with different k varying from 3 to 9. Finally, for fairness, we set the maximal number of dimensions for DPOP to 9.

4.2 Experimental Results

Fig. 3 presents the experiment results under different agent numbers. It can be seen from the figure that PT-FB cannot solve the problems with the agent number greater than 22. That is due to the fact that the search-based solvers need to explicitly exhaust the solution space by message-passing, which is quite expensive when solving the problems with the large agent number. Similarly, given the memory budget, DPOP also fails to scale up to the problems with the agent number more than 22. On the other hand, the scalability of the memory-bounded inference solvers depends on the size of k . For example, MB-DPOP ($k = 3$) can only scale up to the problems with 26 agents while MB-DPOP ($k = 9$) can solve the ones with 30 agents. This is because a large k leads to fewer cycle-cut nodes and can significantly reduce the number of the memory-bounded inferences. Among these memory-bounded inference algorithms, given the same k , RMB-DPOP substantially outperforms MB-DPOP in both communication overheads and runtime. Besides, except for $k = 6$, RMB-DPOP (k) can solve the problems with the larger agent

number than MB-DPOP (k), which demonstrates our proposed mechanisms can improve the scalability of MB-DPOP. It is worth noting that the network load of RMB-DPOP ($k = 3$) is less than MB-DPOP ($k = 6$) when solving the problems with 30 agents, which indicates the merit of our proposed ISM.

Fig. 4 presents the results when solving the problems with different graph densities. It can be concluded from the figure that DPOP can only solve the problems with the density of 0.1 under this configuration. Besides, given the same k , RMB-DPOP (k) outperforms MB-DPOP (k) on all the metrics. Moreover, the gaps between RMB-DPOP and MB-DPOP are widen as the density grows, which demonstrates the potential of RMB-DPOP for reducing redundant inferences. Besides, it is noteworthy that RMB-DPOP with stricter memory budget can still outperform MB-DPOP with relatively large k in terms of network load. For example, the network load of RMB-DPOP ($k = 6$) is even less than the one of MB-DPOP ($k = 9$) when solving dense problems. The same phenomenon also appears in solving the problems with the density of 0.3. The phenomena indicate that the redundant inference in MB-DPOP grows quickly as growing the graph density, while our proposed algorithm can effectively reduce unnecessary inferences. Fig. 5 shows the ratio of the problems successfully solved within different time limits on this configuration where the graph density is 0.5. It can be seen that RMB-DPOP ($k = 9$) solves over 90% of the problems in 15 minutes, while the success rate of MB-DPOP ($k = 9$) is less than 80%. Besides, RMB-DPOP ($k = 9$) solves 60% problems in 6 minutes, while MB-DPOP ($k = 9$) needs another 3 minutes (i.e., 9 minutes) to make that rate, which demonstrates the great superiority of the proposed algorithm again.

Fig. 6 shows the performance comparison when solving scale-free network problems with different m_1 . PT-FB still cannot scale up due to prohibitively large search space. And it is worth mentioning that DPOP fails to solve all these problems, which demonstrates the poor scalability of DPOP under memory-limited scenarios. The reason is because the pseudo trees of scale-free network problems have induced width greater than 9 when $m_1 = 2$. On the other hand, our algorithm exhibits great advantage over MB-DPOP on all the metrics. The results show that RMB-DPOP ($k = 9$) successfully solves all the problems, and except for $k = 6$, RMB-DPOP (k) can solve the problems with larger m_1 than MB-DPOP (k). Although

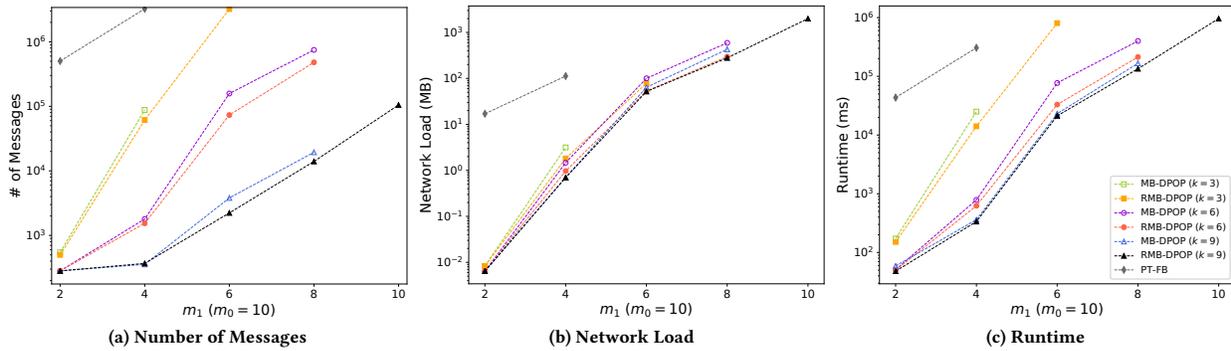


Figure 6: Performance comparison on scale-free networks

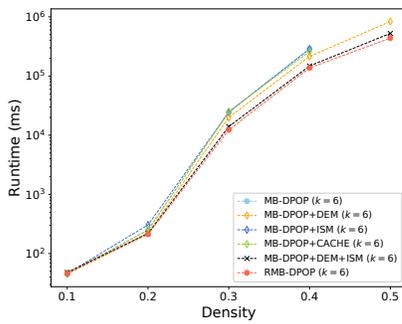


Figure 7: Performance of different mechanisms

the scalability of RMB-DPOP ($k = 6$) seem to be the same with the one of MB-DPOP ($k = 6$), it can be seen that RMB-DPOP ($k = 6$) incurs less network load than MB-DPOP ($k = 9$) and its runtime closes to MB-DPOP ($k = 9$) when $m_1 \geq 6$. It also demonstrates the scalability of our algorithm.

Fig. 7 presents an ablation study on the second configuration with $k = 6$ to demonstrate the effectiveness of each mechanism. It can be seen that the performance of MB-DPOP can be improved by each single mechanism when solving the dense problems, and can be further enhanced via the combinations of DEM and ISM. That is because ISM tends to choose the CC nodes inside clusters and DEM can effectively exploit these CC nodes to reduce the nonconcurrent instantiations. Without DEM and ISM, the contribution of caching mechanism is quite limited, but the combination of all the mechanisms achieves the best performance.

5 CONCLUSIONS

MB-DPOP suffers from a severe redundancy in memory-bounded inference due to the inability of exploiting the structure of a problem. In this paper, we propose a novel algorithm named RMB-DPOP which incorporates three mechanisms to reduce the redundancy in memory-bounded inference. First, we propose a distributed enumeration mechanism to make use of the independence among different branches to reduce the number of nonconcurrent instantiations. Second, we propose an iterative selection mechanism to refine the cycle-cut node selection, which aims to make each cycle-cut node

to cover a maximum of the nodes with the width greater than k in a cluster. Finally, a caching mechanism that exploits the historical inference results is introduced to further avoid unnecessary inferences. We theoretically prove that the distributed enumeration mechanism can reduce the message number if there is at least one cycle-cut node inside the clusters. Our experimental evaluations demonstrate the superiority of RMB-DPOP.

We note that our proposed mechanisms can be adapted to other algorithms as well. In more detail, the caching mechanism could be applied to an iterative process with recurrent combinations. Moreover, the selection mechanism could be used in other memory-bounded inference like ADPOP[21] and HS-CAI[3] to choose more appropriate variables to approximate or decimate. Also, this mechanism is highly customizable when combining with other algorithms. For example, we could easily implement different heuristics by changing the definition of Eff_i for each candidate. Therefore, we envisage that these mechanisms not only advance the development of MB-DPOP, but also contribute to the algorithmic design of DCOPs.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is partially supported by the Chongqing Research Program of Basic Research and Frontier Technology under Grant No.: cstc2017jcyjAX0030, the National Natural Science Foundation of China under Grant No.: 51608070 and the Graduate Research and Innovation Foundation of Chongqing, China under Grant No.: CYS18047.

REFERENCES

- [1] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
- [2] Ismel Brito and Pedro Meseguer. 2010. Improving DPOP with function filtering. In *AAMAS*. 141–148.
- [3] Dingding Chen, Yanchen Deng, Ziyu Chen, Wenxin Zhang, and Zhongshi He. 2019. HS-CAI: A Hybrid DCOP Algorithm via Combining Search with Context-based Inference. *CoRR* abs/1911.12716 (2019). arXiv:1911.12716
- [4] Ziyu Chen, Yanchen Deng, Tengfei Wu, and Zhongshi He. 2018. A class of iterative refined Max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems* 32, 6 (2018), 822–860.
- [5] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1-2 (1999), 41–85.
- [6] Rina Dechter, David Cohen, et al. 2003. *Constraint processing*.

- [7] Peibo Duan, Changsheng Zhang, Guoqiang Mao, and Bin Zhang. 2018. Applying Distributed Constraint Optimization Approach to the User Association Problem in Heterogeneous Networks. *IEEE Trans. Cybernetics* 48, 6 (2018), 1696–1707.
- [8] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*. 639–646.
- [9] Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar J Ranade. 2017. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In *AAMAS*. 999–1007.
- [10] Eugene C. Freuder and Michael J. Quinn. 1985. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *IJCAI*. 1076–1078.
- [11] Amir Gershman, Amnon Meisels, and Roie Zivan. 2009. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research* 34 (2009), 61–88.
- [12] Patricia Gutierrez and Pedro Meseguer. 2012. Removing redundant messages in n-ary BnB-ADOPT. *Journal of Artificial Intelligence Research* 45 (2012), 287–304.
- [13] Patricia Gutierrez, Pedro Meseguer, and William Yeoh. 2011. Generalizing adopt and bnb-adopt. In *IJCAI*. 554–559.
- [14] Katsutoshi Hirayama and Makoto Yokoo. 1997. Distributed partial constraint satisfaction problem. In *CP*. 222–236.
- [15] Shijie Li, Rudy R. Negenborn, and Gabriël Lodewijks. 2016. Distributed constraint optimization for addressing vessel rotation planning problems. *Engineering Applications of Artificial Intelligence* 48 (2016), 159–172.
- [16] Omer Litov and Amnon Meisels. 2017. Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence* 252 (2017), 83–99.
- [17] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In *ISCA PDCS*. 432–439.
- [18] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161, 1-2 (2005), 149–180.
- [19] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. 2019. Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm. *Journal of Artificial Intelligence Research* 64 (2019), 705–748.
- [20] Jonathan P Pearce and Milind Tambe. 2007. Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems. In *IJCAI*. 1446–1451.
- [21] Adrian Petcu and Boi Faltings. 2005. Approximations in distributed optimization. In *CP*. 802–806.
- [22] Adrian Petcu and Boi Faltings. 2005. A scalable method for multiagent constraint optimization. In *IJCAI*. 266–271.
- [23] Adrian Petcu and Boi Faltings. 2006. ODPOP: An algorithm for open/distributed constraint optimization. In *AAAI*. 703–708.
- [24] Adrian Petcu and Boi Faltings. 2007. A hybrid of inference and local search for distributed combinatorial optimization. In *ICLAT*. 342–348.
- [25] Adrian Petcu and Boi Faltings. 2007. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. In *IJCAI*. 1452–1457.
- [26] Evan Sultanik, Pragnesh Jay Modi, and William C Regli. 2007. On Modeling Multiagent Task Scheduling as a Distributed Constraint Optimization Problem. In *IJCAI*. 1531–1536.
- [27] William Yeoh, Ariel Felner, and Sven Koenig. 2010. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research* 38 (2010), 85–133.
- [28] William Yeoh, Xiaoxun Sun, and Sven Koenig. 2009. Trading off solution quality for faster computation in DCOP search algorithms. In *IJCAI*. 354–360.
- [29] William Yeoh, Pradeep Varakantham, and Sven Koenig. 2009. Caching schemes for DCOP search algorithms. In *AAMAS*. 609–616.
- [30] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. 2005. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161, 1-2 (2005), 55–87.